(2)

# An Expert System for Searching in Full-Text

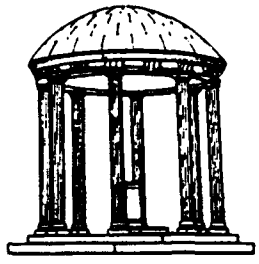
*TR89-043*

*December, 1989*



*Susan Gauch*





The University of North Carolina at Chapel Hill
Department of Computer Science
CB#3175, Sitterson Hall
Chapel Hill, NC 27599-3175

**A TextLab Report**
*This work was supported in part by ONR contract N00014-86-K-0680.*

# An Expert System for Searching in Full-Text

by

Susan Evalyn Gauch

A dissertation submitted to the faculty of The University of North Carolina at Chapel Hill in partial fulfillment of the requirements for the degree of Doctor of Philosophy in the Department of Computer Science.



Chapel Hill
1990

Approved by

Advisor: Dr. John B. Smith

Reader: Dr. Stephen Weiss

Reader: Dr. Judith Wood

## Abstract

This dissertation explores techniques to improve full-text information retrieval by experienced computer users who are novice users of retrieval systems. An expert system which automatically reformulates Boolean user queries to improve search results is presented. The expert system differs from other intelligent database functions in two ways: it works with semantically and syntactically unprocessed text; and the expert system contains a knowledge base of domain independent search strategies. The passages retrieved are presented to the user in decreasing order of estimated relevancy. This combination of user interface features provides powerful, yet simple, access to full-text documents.

Experimental results demonstrate that the expert system can improve the search efficiency of novice searchers without decreasing their search effectiveness. Further, an evaluation of the ranking algorithm confirms that, in general, the system presents potentially relevant passages to the user before irrelevant passages.

## Acknowledgments

Many qualities are required of a doctoral candidate to enable them to reach their goal. They need intelligence to pass the required exams, naivete when choosing their thesis topic, and persistence to complete the research. More than this, they need the unflagging support of family, friends, and colleagues when the final year inevitably drags on for two.

At this time I would like to thank my parents for the intelligence I was born with, and the love of learning they instilled afterwards. They are unable to share this accomplishment with me, but they are always in my thoughts.

My committee (Frederick P. Brooks, Jr., David Plaisted, John B. Smith, Stephen Weiss, and Judith Wood) deserves a chorus of praise for doing their best to offset my naivete. Their insistence on a firmly rooted proposal guided me to an achievable research plan. Cutting the proposal in half allowed me to graduate this decade.

Persistence comes from within, but external encouragement is necessary to get over the rough spots. I owe a great debt to my advisor John B. Smith for his enthusiasm every step of the way.

Finally, I want to thank my husband John Gauch for his technical, emotional, and financial support. Focusing on this dissertation after the birth of our daughter Laura was difficult, but it would have impossible without John's help. He is therefore forgiven for defending his dissertation before me.

## Table of Contents

# *Chapter 1*
# *Introduction*

## 1.1 Driving Problem

Technological advances are causing a revolution in information retrieval. Optical character recognition, word processors, and computer publishing software are capable of producing massive quantities of online text. The development of optical storage media is making the storage and distribution of large collections of online text feasible. Proliferation of personal workstations, combined with modems, are allowing an increasing number of end-users to do their own searching of online databases. *Textbases*, online full-text databases, are becoming more common. BRS, a major vendor of online databases, already provides access to over twenty full-text databases compared with just one in 1981. All these trends lead to end-user's performing their own textbase searches.

The main roadblock to wide-spread use of online textbases will soon be the inability of end-users to search effectively. Christine Borgman [Borgman, 1986] identifies two types of knowledge necessary to search: knowledge of the mechanical aspects of searching (e.g., syntax and semantics of both the query language and the system interaction commands) and knowledge of the conceptual aspects (e.g., ways to broaden and narrow searches using alternative vocabulary, choosing alternative search paths). She summarizes the results of many different user studies, concluding that, whereas system mechanics are rarely a problem for any but very inexperienced and infrequent users, even experienced searchers have significant problems with search strategy and output performance.

User difficulty with search strategy shows up in many different studies on searching online bibliographic databases. Carol Fenichel [Fenichel, 1981] finds that even experienced searchers could improve their search results. The searchers lost sight of the search logic, missed obvious synonyms, and searched too simply. Search performance is often measured by recall, the ratio of relevant documents retrieved to the number of relevant documents in the entire database. The searchers were satisfied with 51% recall on average, indicating that almost half of the relevant information was not retrieved. The lack of successively refining queries, called *iteration*, is another problem identified. In spite of the low recall, half of the searchers never modified the original query in an attempt to improve their results.

Studies of inexperienced searchers find even more problems with search strategy. In one study [Borgman, 1987], a quarter of the subjects were unable to pass a benchmark test of minimum searching skill. In another experiment [Oldroyd, 1984], contrasting the searching of novices versus experienced searchers, the novices found some relevant documents easily, but they failed to achieve high recall and were unable to reformulate queries well. The experienced searchers in this study were more persistent and willing to experiment than the novices.

David Blair [Blair and Maron, 1985] paints an even bleaker picture for searching full-text databases. Lawyers searching a legal database achieved only 20% recall, although they were attempting to do a high recall search. The factors, as identified by the authors, leading to this poor performance were poor searching technique (failure to use stemming and

synonyms), stopping the query iteration too soon, and the inability to search on inter-document relationships.

## 1.2 Research Overview

To address the problems identified above, I have developed a query reformulation expert system to act as a front-end to a textbase. The expert system addresses the problems of too little iteration and improper use of vocabulary identified in several different user studies. This prototype augments the searching capabilities of novice users by providing automatic query reformulation to improve the search results and automatic ranking of the retrieved passages to speed the identification of relevant information. The hope is that, given such a tool, novice searchers will be able to search more efficiently and more effectively. The prototype was tested under experimental conditions to demonstrate the feasibility of an online intelligent assistant function.

The system uses the MICROARRAS system [Smith et al, 1987] to provide full-text search over the entire contents of the experimental textbase. A knowledge base of search strategies guides the automatic query reformulation, similar to PLEXUS [Vickery and Brooks, 1987] and the OCLC system [Teskey, 1987], although my collection of search strategies is more extensive. The knowledge base is based on the published literature of studies of expert searchers, search training manuals, consultation with professional searcher, and my own searching experience.

Domain knowledge is supplied by a predefined online thesaurus, similar to IOTA [Chiaramella and Defude, 1987] and IR-NLI II [Brajnik et al, 1988]. Since the domain knowledge is separated from the searching knowledge base, the search strategies are domain-independent. This should allow the expert system to be applied to a different domain by merely replacing the textbase and thesaurus.

Finally, the system includes a ranking function. Jung Soon Ro's [Ro, 1988] work comparing ranking schemes formed the basis of the ranking algorithm used. His algorithms were modified to work with a single text database and to incorporate the contextual information available to the expert system.

## 1.3 Related Work

Research to improve access to online information is proceeding in many directions. By helping the user with the mechanics of their search, and by providing access to an online thesaurus, better user interfaces can lead to improved search results with existing databases. Some projects in this field are summarized in Section 1.3.1. Similarly, allowing the user to query the database in his natural language, rather than requiring him to form a Boolean query, may lead to simpler searching. Some natural language interfaces are described in Section 1.3.3.

Accessing relevant information may also become easier by improving the quality of the information that is stored in the database. The three main approaches are: representing documents and search term as an associative network (see Section 1.3.2); using natural language processing techniques to select index terms (see Section 1.3.3); and building a knowledge base from the document contents. Based on the contents of the knowledge base, the user receives direct answers to his queries, rather than documents or document passages.

Finally, search performance may be improved by providing an online search assistant as the front-end to existing retrieval systems. Research in this area is summarized in Section 1.3.4. The knowledge bases for these systems, including my own, should be built on existing searching practice. Current knowledge on good search technique is presented in Section 1.3.5.

### 1.3.1 User Interfaces

As the demand for direct access to existing online information retrieval systems has grown, so has interest in providing friendlier interfaces. Marcus [Marcus, 1981] and Meadow [Meadow et al, 1982] describe research prototypes based on conventional programming techniques which make existing bibliographic databases easier to search. These projects focus on providing menu systems to guide novice users. The menus provide information about the different databases available, selecting search terms, and connecting to a remote database. More recently, Richard Marcus [Marcus, 1986] has proposed an online help facility which would include a menu of query reformulation techniques.

The F-TAS system [Prasse et al, 1988] provides a state of the art menu/mouse/icon based user interface to a full-text retrieval system. It has a state-of-the-art user interface, requiring user terminals with graphic abilities, and is designed specifically for searching in an online book. The user searches by accessing the book's table of contents, back of the book index, or by doing a free-text search.

Essentially, these systems aid in the mechanical aspects of searching, with less emphasis on helping users with the more significant conceptual problems of searching.

H. P. Giger [Giger, 1988] has built a user interface incorporating an online thesaurus. The user enters a Boolean query which retrieves a set of documents. The user's search terms are located in the thesaurus, from which sub-concepts and overlapping concepts can be identified. These related concepts are then used, with the original search terms, to rank the retrieved documents. No query reformulation is attempted.

Many projects have looked at the possibility of allowing users to query databases in natural language, removing the need for them to form Boolean queries. Since these projects usually incorporate some natural language processing of the database as well, they are discussed in Section 1.3.3.

### 1.3.2 Associative Networks for Information Retrieval

My approach is to improve search performance by providing an expert system front-end to existing textbases. An alternative approach to improve search performance is to change the information that is stored in the database. Artificial intelligence research into knowledge representation has spawned investigations into using semantic or associative networks to implement information retrieval systems. The hope is that, by providing links between related documents and terms in the database, relevant information will be easier to find, and irrelevant information easier to discard.

One type of associative network that can represent a database is one based on citations and references. Bruce Croft [Croft et al, 1988] has built such a network. He evaluates several techniques for locating the most closely related nodes in the network. The highest ranking 20 documents from a standard probabilistic search are used as the initial nodes, and several techniques for using the link information in the network were tested. The best technique seems to be quite good at locating relevant documents, at low recall levels.

THOMAS [Oddy, 1977] is a system containing an associative network with nodes for authors, documents, and terms. Documents are linked to documents they reference or are cited by, to the terms used to index them, and to their authors. The user enters a set of terms which are used to locate the most "highly involved" document. The user is expected to browse through the connected nodes on his own. Robert Ledwith [Ledwith, 1988] is building a semantic network of documents to allow user browsing through the Chemical Abstracts literature. Using the available taxonomic information as his domain knowledge, he has built a large (over 900,000 nodes) semantic network. A short description of each document is parsed, and the document is linked into the existing semantic net. It will be interesting to see how effectively users can navigate through such a complex structure.

AIR [Belew, 1989] is a more sophisticated associative network system. The network contains document, term, and author nodes as in THOMAS. The user enters an initial Boolean query which selects, or *activates,* some of the term nodes. Each node that is reachable from an activated node is assigned a weight. The weight a node receives decreases as the number of links traversed from the initially activated nodes increases. The weight increases with the number of initially activated nodes from which it can be reached. The propagation of the weights along the links in the network is called *spreading activation.* The documents are ranked according to the total weight they receive. AIR has the added feature that the links between nodes receive weights based on the user's relevance feedback.

Gerard Salton [Salton and Buckley, 1988] compared a simple spreading activation model to standard term weighting schemes. Whereas precision seems to be slightly worse for the activation spreading formulae he used, he felt that a more sophisticated form of activation spreading could lead to improved results.

In conclusion, search performance may be improved by changing or augmenting the information stored in databases to include associative links. More research must be done to demonstrate that this approach does lead to improved performance before it is included in commercial databases. Even if associative networks do provide improvements, the cost of improving search performance by providing an intelligent front-end is likely to be less than approaches based on changing the contents of the databases themselves.

### 1.3.3 Natural Language Processing

*Syntax*

Several projects have attempted to include syntactic processing into the retrieval process. A popular approach is to parse the documents in the database to identify multi-word phrases which are then used to index the documents. Gerard Salton [Salton and Smith, 1989] compares the sets of syntactically identified phrases with those generated statistically, as candidates for indexing a book. Using a state-of-the-art parser from IBM, he finds that one third of the sentence parses were seriously flawed, leading to more incorrect phrases and more missed correct phrases than those generated statistically. However, several recent improvements in parsing may lead to better results in the near future.

Other syntactically based systems allow the users to query the database in their natural language. The queries are then parsed to locate hierarchical relationships between terms in the query, called *dependencies*. These dependencies are searched for in the text of the online documents. Douglas Metzler [Metzler and Haas, 1989] describes a parser which is capable of reliably identifying such dependencies, although some sentences generate more than one possible parse. Furthermore, A. F. Smeaton [Smeaton and van Rijsbergen, 1988] found that ranking documents based on dependencies occurring only in noun phrases lead to improved precision.

*Semantics*
Semantically based natural language processing is used for two different purposes: to improve the performance of retrieval systems, and to build knowledge bases which are used for question-answering systems. Some examples of the former include: ranking algorithms which replace anaphora with the noun phrases to which they refer [Bonzi and Liddy, 1988]; and indexing documents with word senses, identified using an online dictionary, rather than words [Krovetz and Croft, 1989].

A hybrid system is being built by Bruce Croft [Croft and Lewis, 1987]. Full natural language understanding is attempted on the user queries, causing a frame to be filled in. The information in the frame is used to perform a statistical search and to rank the retrieved documents. High-ranking documents are presented to the user, and natural language processing is used to re-rank the medium-ranked documents.

A different type of information system, called a *question-answering* system, has received much attention in the artificial intelligence world. Initially, knowledge bases were constructed by hand, and users were allowed to query the knowledge base and receive their answers in natural language [Zarri, 1984a] [Zarri, 1984b]. BASEBALL [Green et al, 1969] was one of the earliest question-answering systems. It answered English language questions about a database of baseball facts. Due to the limited domain of discourse, the restricted form of user queries, and the use of deduction, BASEBALL produced impressive results.

More recent projects have focused on automatically constructing the knowledge base from natural language text [Katz, 1988]. Some examples of question-answering systems which build their knowledge bases from input text are Researcher [Lebowitz, 1985], which has a knowledge base of patents, and SCISOR [Jacobs and Rau, 1988], which collects merger and takeover information from newspaper stories. Building a knowledge structure from natural language text is a slow and error-prone process that is currently not feasible for large, dynamic collections of documents. Even if knowledge structures can be built and queried effectively for large document collections, many users will want to see the actual text of the original documents, not just a synthesized answer to their query. I agree with Karen Sparck Jones that "The language of documents is part of their information content." [Sparck Jones, 1983].

### 1.3.4 Expert Systems

The exploration of possible applications of expert system techniques to information retrieval systems has generated interest. Early projects are surveyed by Karen Sparck Jones [Sparck Jones, 1983] and Helen Brooks [Brooks, 1983], whereas Nicholas Belkin [Belkin et al, 1987] summarizes more recent research. The most common goal is to develop an expert system to help with the retrieval process by assuming some of the tasks of the search intermediary. An exception is James Driscoll's [Driscoll et al, 1988] expert system whose task is to index documents. This section will give an overview of expert system projects designed for bibliographic retrieval and those which work with full-text databases.

*Bibliographic Expert Systems*
Steven Pollitt [Pollitt, 1984] has built one of the earliest expert systems for bibliographic retrieval. It is designed to search the MEDLINE medical database for cancer literature. The expert system can search cancer literature only, since the knowledge base contains information on cancer, rather than on search strategies in general. Whereas the performance of the system has not been formally analysed, informal evaluations have been

discouraging. The project has since been replaced by CANSEARCH [Pollitt, 1987], an expert system to guide users in the use of menus to form their own queries.

IR-NLI II [Brajnik et al, 1988] are incorporating user modeling into a domain-independent bibliographic retrieval expert system. Domain knowledge is supplied by an online thesaurus. A user model is built based on the user's amount of domain knowledge and search experience. This model is used to tailor the dialogue between the system and the user. Initially, the user lists some terms which describe his interests. The expert system, through a lengthy dialogue, clarifies its model of the query, proposes terms to expand the query, and comments on the user's search strategy. No automatic query reformulation is done.

IOTA [Chiaramella and Defude, 1987] is a bibliographic expert system which incorporates a natural language interface. Whereas the expert system does passage retrieval from an online book, I include the system with the bibliographic systems because retrieval is done on keywords which index each passage. Much of the research effort has gone into processing the user's queries, but some simple query reformulation is also done. Specifically, queries are broadened by replacing a term by its parent from an online thesaurus and narrowed by removing OR terms. Their results show an increase in precision and recall using the expert system. These results are tentative, since the textbase is very small (3,000 words), the thesaurus is small (118 classes), and only 12 queries were run.

PLEXUS [Vickery and Brooks, 1987] is an expert system to help novice users find information about gardening. Natural language queries are accepted, and information is extracted to fill in frames. If a frame is too incomplete, the user is asked for more information. Once the frames contain enough information, a query is sent to the online database. The system has a knowledge base of search strategies and term classifications similar to a thesaurus. Most of the domain knowledge is in the classification, but some appears in the rule base, itself. If queries are too broad (defined as more than 10 references) , no narrowing is attempted. The references are displayed 5 at a time to the user. If the query is too narrow (defined as nothing retrieved at all), three strategies are attempted: 1) if two or more terms appear in the same subcategory, OR them together rather than AND; 2) drop one of the terms; 3) replace a term by its parent. There has been no published evaluation of the system's performance.

*Full-Text Expert Systems*

There have been fewer projects aimed at providing intelligent assistance for full-text searching. The earliest such system is RUBRIC [McCune et al, 1985] [Tong et al, 1987], which has the user describe his query in terms of rules. These rules describe the domain knowledge for the system as a hierarchy of topics and subtopics. Rules may have weights representing the certainty and/or importance of the defined relationships. The lowest level subtopics define patterns in the text which indicate the presence of that subtopic. Whereas the query language is very powerful, it places a heavy burden on the user.

I$^3$R [Croft and Thompson, 1987] also requires the user to provide the appropriate domain knowledge. The query process is a dialogue between the user and the system, during which the user is asked to supply a semantic network, or thesaurus, that describes the relationships among the concepts in his query. A blackboard architecture is employed to control the search process, which consists of the user dialogue, probabilistic search, cluster based search, and user feedback. There is no automatic query reformulation, but the expert system is able to infer the presence of concepts from the presence of related concepts described in the domain knowledge structure.

A full-text system that incorporates query reformulation assistance is under development at OCLC [Teskey, 1987]. The emphasis to date has been on providing an intelligent online help function, but a few basic reformulation strategies are provided. Queries are broadened by asking the user to OR together ANDed concepts, or to drop a concept altogether. Narrowing is suggested when a single broad search term retrieves more than 30 passages. If this happens, the system first searches for multi-word phrases containing the term in the back-of-the-book index and the table of contents. These phrases, if found, are presented to the user as alternate queries. If no such phrases are found, the system returns the passages which are clustered. If there is no clustering of hits, a random selection is shown.

The CODER system [Fox et al, 1988] is an ambitious project that incorporates natural language processing with expert systems techniques to produce a testbed for evaluating advanced information retrieval techniques. It is designed for the domain of electronic mail messages. The expert system identifies the structure within a message and semantic relationships between messages. There is no user assistance, per se.

### 1.3.5 Search Strategies

The automatic query reformulation incorporated in the systems described in the previous section are, in general, very primitive. However, search strategies employed by both novice and experienced searchers have been widely studied. These studies formed the basis of my expert system's searching knowledge base, which is described in detail in Chapter 3.

*Searching Studies*
The most thorough catalogue of search tactics was compiled by Marcia Bates [Bates, 1979]. She outlines 29 search tactics in four areas: monitoring, file structure, search formulation, and term manipulation. The tactics for search formulation and term manipulation describe the available techniques to broaden and narrow queries. The search formulation tactics include the selection of appropriate initial search terms and the manipulation of query structure; the term manipulation tactics describe the use of context, thesaural terms, and stemming to modify queries. The tactics she lists provide the basic operations for my expert system; however, she includes no guideline as to when each tactic is appropriate. Bates concludes by saying that knowing when to stop a search is a difficult problem.

Philip Smith [Smith et al, 1989] reports on a similar study as the first step to building an online search intermediary for searching the environmental literature of Chemical Abstracts. By analysing the discourses and actions of 17 users and search intermediaries, he compiles a list of 19 search tactics. He believes that the search tactics he finds are domain-dependent, but I tend to disagree. The majority of the tactics finds are similar to those described by Bates [Bates, 1979]. Domain knowledge is involved only in selecting which terms to include when there were too many candidates from the thesaurus. This selection of terms from a set of candidates is called *filtering*.

P. W. Williams [Williams, 1984] has developed a model of all possible search situations and all possible responses, to be used as an expert system's knowledge base. Based on the desired, versus the achieved, values of three variables (number of documents, precision, and recall), he identifies 64 search situations which result in 27 unique states. He defines four variables (generality, exhaustivity, simplicity, ambiguity) which can be manipulated to respond to each state in an attempt to achieve the desired search results. Although he describes some techniques to manipulate the four variables, he does not indicate how the techniques should be combined or when they should be applied. In addition, several of the

states have conflicting demands which are hard to resolve. It is an interesting categorization of searching situations, but it is not yet developed enough to become the basis of an automatic search assistant.

*Effects of Query Expansion*
A. F. Smeaton [Smeaton and van Rijsbergen, 1983] has studied the effects of query expansion on retrieval performance. He finds that automatically adding terms based on their statistical relationships to the user's search terms degrades retrieval performance. He argues the need for better criteria for selecting terms to add. Donna Harman [Harman, 1988] also shows performance degradation when adding terms from a statistically constructed thesaurus. However, when only those thesaural terms which occur in documents already flagged as relevant by the user are added, retrieval performance improves over that achieved by the original query. Allowing the user to add variants of the original search terms to the queries proves to be better than selecting from thesaural terms. However, statistically selected terms from the relevant documents proves to be the best candidate for query expansion. Finally, the best performance is achieved when user filtering of the three types of candidate terms (thesaural, term variants, and statistically selected from relevant documents) is simulated.

Carolyn Crouch [Crouch, 1988] has investigated the use of terms from an automatically constructed thesaurus for query reformulation. She concludes that augmenting a query with thesaurus terms, rather than replacing the user's original search terms, improves performance. She also advises that, for document ranking, terms included from a thesaurus should receive lower weights.

### 1.3.6 Summary

This chapter has given an overview of some of the research involving information retrieval systems. Exploratory systems incorporating sophisticated user interfaces provide assistance with search mechanics. Whereas improved interfaces are desirable, search strategy, not mechanics, is the main source of difficulty for searchers.

Other research systems have focused on the use of associative networks as a representation for the database. These have not yet been shown to provide any improvements in search results over inverted file systems, and it is not clear how they could be extended to full-text applications.

Systems which include syntactic processing may improve indexing or query processing, if more reliable parsers can be developed. However, they provide no assistance with search refinement. Furthermore, systems which apply semantic pre-processing to the text seem unlikely to be able to handle large online databases in the near future. For the near term, I believe that retrieval breakthroughs will come from developing online intelligent assistants to allow novice users to search easily and adequately.

## 1.4 Research Goals and Results

### 1.4.1 Goals

The goals of this research are to develop a prototype, as described in the Section 1.2, and to evaluate its performance experimentally. The system's effectiveness and efficiency, two measures of performance described by Gerard Salton [Salton & McGill, 1983], need to be evaluated. Basically, the effectiveness of an information system is a measure of the system

performance, whereas efficiency is a measure of the amount of user effort required to perform a task. My goals, then, are the following:

Goal 1: Develop an expert system for query reformulation.

Goal 2: Demonstrate that the query reformulation expert system improves the search effectiveness for a novice searcher [Hypothesis 1].

Goal 3: Demonstrate that the query reformulation expert system improves the search efficiency for a novice searcher [Hypothesis 2].

Goal 4: Demonstrate that an expert system can rank the passages retrieved by the search in decreasing order of relevance [Hypothesis 3].

Search effectiveness is measured by the number of relevant passages retrieved and the standard information-retrieval measures of precision and recall. Precision is defined as the ratio of the number of relevant documents retrieved to the total number of documents retrieved. Recall is defined as the ratio of the number of relevant documents retrieved to the total number of relevant documents in the database. Search efficiency is measured by the number of queries and the amount of time necessary to perform the search. These measures are calculated and reported separately. Together, they give an overview of the effects of providing an online search assistant.

### 1.4.2 Summary of Results

The first goal, the development of a prototype expert system for searching in full text, was achieved. This system was evaluated experimentally. Twelve computer science graduate students queried three configurations of a full-text retrieval system: 1) the subjects queried on their own, using a contextual Boolean query language; 2) the subjects queried as in 1), but they had access to an online thesaurus; 3) the subjects queried with the help of the online expert system. Each subject performed four monitored queries with each system. The order of presentation of the systems, and the queries performed with each system, were counterbalanced among the subjects. The statistically significant results of the experiment are summarized here. For a full discussion of the experiment, and its results, see Chapter 4.

The goal of improving the search effectiveness of novice searchers was partially demonstrated with this experiment. The experiment asked subjects to achieve a specific level of effectiveness and measured their efficiency. The expert system was able to achieve an improvement in precision. A statistically significant difference was found between the 60.4% precision achieved with the expert system versus 53.0% for the users searching on their own. The 57.6% precision achieved by the user working with the thesaurus was not significantly different from either of the other two means. There was no significant improvement in recall. However, since it took the subjects more queries to retrieve the desired number of passages working without the expert system, it follows that after a set number of queries they had lower recall without the expert system.

The most important result of this research was the achievement of the third goal, proof that an expert system could improve the search efficiency of novice searchers. The expert system significantly decreased the number of queries the users required to retrieve approximately ten relevant passages (see Section 4.3.2). The user effort was cut to less than half, from an average of 4.83 queries per question when the subject searched alone (5.46 queries using the thesaurus) to only 2.35 queries per question using the expert

system. Thus, the expert system enabled human searchers to search effectively with half the effort.

The final goal, demonstrating that the expert system was able to rank order the retrieved passages, was also achieved. Relevant passages were presented earlier in the retrieval set than would be expected due to randomness. The normalized balance point for the ranked passages was 0.195 (on a scale from -1 to +1), compared with a balance point of 0 produced by a random ordering. The improvement was slight, but significant.

The main result of this research is to demonstrate that an expert system can make searching easier for novice searchers. This is important because it increases the usefullness of online full-text databases. If more people can successfully search online textbases, and they can do so with less effort, the information stored in these textbases will become more widely disseminated.

# *Chapter 2*
# *System Architecture*

## 2.1 Overview

The prototype system consists of five major components (see Figure 2.1):

1) MICROARRAS, which serves as the full-text search and retrieval engine
2) a full-text database of over 188,000 words
3) a hierarchical thesaurus of approximately 7,424 words specific to the textbase's domain
4) an expert system of 85 OPS83 rules and over 5,000 lines of C code, which interprets the user's queries, controls the search process, analyzes the retrieved text, and ranks the search results
5) a user interface, which accepts the user's queries, presents requests for information from the expert system, and displays the search results.

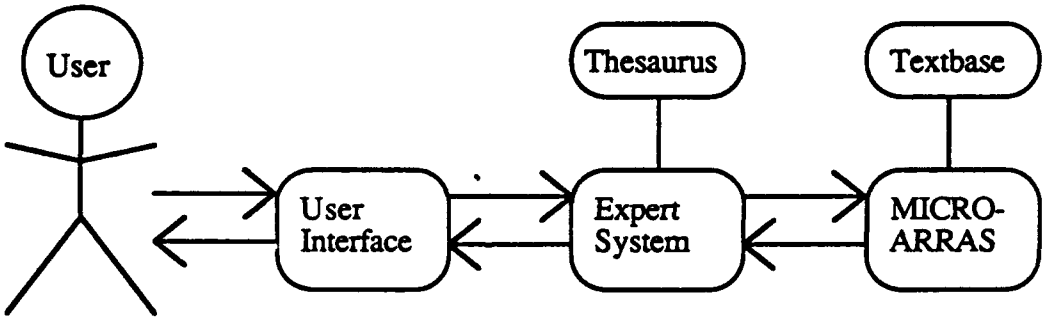



Figure 2.1. System Architecture

The system is implemented on a Sun 3 workstation. MICROARRAS and the thesaurus construction and access routines are written in the C language. The expert system consists of a knowledge base of production rules, written in OPS83, and a set of C language functions to carry out the actions prescribed by the rule-base. The textual database for the current demonstration project consists of an unpublished manuscript on computer architecture written by Gerrit A. Blaauw and Frederick P. Brooks, Jr. [Blaauw and Brooks, 1987]. The search process consists of a dialogue between the user and the expert system. The user enters the initial Boolean query and the number of passages he would like to retrieve. The expert system parses the query and translates it into a request for information from MICROARRAS. MICROARRAS retrieves text passages from the full-text database and informs the expert system of the number of passages that satisfy the request. The expert system compares the number retrieved with the target number to decide whether or not to reformulate the query, and, if so, how.

To expand a search query, the expert system may use three different strategies, alone or in combination. First, it can expand individual search terms to the sets of words using the thesaurus. The words are added iteratively, in the following order: words with the same stem; synonyms; words which are broader in scope; narrower words; and, finally, similar words. Second, it can relax contextual constraints. Since MICROARRAS provides

considerable generality in terms of segmental contexts, search expressions may contain contextual parameters expressed in terms of any number of words, sentences, paragraphs, etc. to either the right or left of any term in the search expression. For example, the context for the query 'architecture AND design' may be broadened from the default context of the same sentence to a context in which the terms must appear, in either order, within three sentences. Finally, it can change the Boolean operators, making the query less restrictive by replacing ANDs with ORs or removing ANDNOTs.

To restrict a search, the expert system uses the same strategies as those described above, but in reverse. That is, it may add sets of search terms to those terms to be excluded from the retrieval passages, contract contexts, and replace ORs with ANDs. Changing the Boolean operators in this way will reduce the number of passages retrieved; however, it is only likely to be useful when the user has used the incorrect Boolean operator in the original query.

Once an appropriate number of passages is identified, the expert system attempts to rank the passages in terms of probable relevance. It does this by performing a rudimentary content analysis on the passages retrieved by MICROARRAS and computing a relevance index for each. The relevance index for each passage is a function of the number of search terms actually found in that passage, the number of distinct types for each (for terms that are sets), and the number of different thesaural categories represented. Query structure, distance between search terms, and frequency of the search terms in the textbase as a whole are also taken into consideration. The retrieved passages are then sorted by their relevance indices and presented to the user in order of probable interest.

A major advantage of this architecture is the separation of strategic knowledge, contained in the knowledge base for the expert system, from domain knowledge, contained in the thesaurus. Now that the search strategy rules have been developed and tested with the existing textbase, the expert system can be tested with other content domains by simply providing a suitable thesaurus for the new textbase.

## 2.2 MICROARRAS

This section describes MICROARRAS, the retrieval software. MICROARRAS developed at the University of North Carolina under the direction of John B. Smith and Stephen Weiss [Smith et al, 1986]. It was used without modification and is not part of this research. However, it is described so that the reader will know which functions, required by the expert system, were provided by the retrieval system.

### 2.2.1 Capabilities

MICROARRAS is an advanced full-text retrieval and analysis system. The system provides direct access to any passage in the textbase. Users can browse through a document's vocabulary as well as its text. MICROARRAS also provides contextual Boolean search on any word or set of words in the text, and can calculate various numeric distributions.

*Textbase*
To be inserted into MICROARRAS's textbase, a document must first be inverted. Inverted files contain a dictionary with one entry for every unique word, called a *type*, and an associated list of all the positions in the text, called *tokens*, of the occurrences of the type. There is no syntactic or semantic pre-processing necessary.

Once the documents have been inverted, they can be examined individually or in groups. They can also be moved from one textbase to another. Thus, documents can be processed on a workstation or microcomputer, uploaded into a textbase on a mainframe or textbase server, searched and analyzed there, or downloaded once again for local use. MICROARRAS can work with a textbase of many different documents, but I will confine this discussion to textbases containing one document only since this is the way it was used.

*Segments*
One of MICROARRAS's strengths is its flexibility in handling context specifications. The system generally views a document as composed of two overlapping hierarchies of text segments. The first denotes logical divisions, such as volume, chapter, paragraph, sentence, and word within sentence; the second, physical divisions, such as page, line and word within line. When a new textual database is established, the segments to be known throughout the database are defined. In addition to these canonical segments, ad hoc segmentations may be used for individual documents, but the user must define any hierarchical relations among them that are to be recognized. These segmentation schemes are used to format the display and to specify contexts for searching.

*Text Display*
MICROARRAS can display all occurrences of a given word along with some designated context around each. Essentially, this is an interactive concordance or keyword-in-context (kwic) listing. MICROARRAS can produce such displays immediately, no matter how large the document or how scattered the occurrences. Contexts for searches can be indicated in terms of words, sentences, paragraphs, etc., for the entire search expression or for different parts of it.

*Lexical Display*
MICROARRAS can access an alphabetical lexicon of the words that appear in a document or passage. Lexical information may be returned in one of three ways: by alphabetic sequence, by pattern of characters, and by frequency of occurrence in the document or passage.

*Categories*
One particularly important feature for this project is a generalized categorization option by which one may define a *category* which contains groups of words. MICROARRAS supports three types of categories: token lists, type lists, and recursive lists. *Token lists* are sets of text positions representing individual occurrences of words; for example, the specific places in a document where *processor* refers to a computer rather than a food processor. *Type lists* represent sets of word types. When they are evaluated for a particular passage, they yield a token list containing each instance in the passage of the members of the type list. For example, the type list *cpu-category* may contain *cpu* and *central processing unit*. Evaluating *cpu-category* with the Blaauw and Brooks textbase will create a token list containing all occurrences of either *cpu* or *central processing unit* in the textbase. *Recursive categories* are sets of other categories; for example, a new category consisting of the *cpu-category* and the *memory-category*.

*Search*
To conduct a search, the user must specify three components: a Boolean contextual expression, a passage in which to evaluate the expression, and a category in which to store the resulting set of positions where the expression occurs (a token list). The expression is any Boolean combination of words or category names. If categories are used, they imply every occurrence of any word type included. For example, the query 'memory-category and (within the same sentence) location' can be evaluated against a particular textbase, or

part of a textbase. The result will be a token list of occurrences of members of memory-category which appear in the same sentence as the word type *location*.

Contexts in which search expressions are evaluated can be specified in terms of any text segment units valid for the textbase and can be specified in any number. Thus, one can look for all occurrences of 'cpu-category and memory-category' within, say, three words of one another, or within the same sentence, or within three sentences, or in the same sub-section, etc. Different contexts can also be specified for sub-expressions, e.g. 'cpu-category AND (within three words) memory-category ANDNOT (within the same sentence) hardware-category'.

*Arithmetic Functions*
MICROARRAS provides facilities to compute various textual measures, display the results, or pass them to an external statistical analysis package. Two kinds of data can be computed. The first is frequencies of words or categories within a specified passage. Thus, the user can compute and display the frequency with which a word or category appears in a paragraph, a chapter, or the whole textbase. The second major class of statistical data are segmental measures. These determine the number of times one segmental measure occurs within another. For example, the number of words within a paragraph, or the sentences in a document. These measures can be stored as scalar or vector numeric variables of integers or reals which can be later analyzed by a statistical package.

### 2.2.2 Implementation

*Document Flow*
Preparing a document for use by MICROARRAS is a four-step process. First, it must be transcribed into a machine-readable form with internal marks indicating logical segments (such as chapters, sections, and paragraphs) and physical segments (line and page). Second, a correspondence is made between the segment names used in the document and the set of segment names used in the textbase as a whole, the *canonical* set of segments. For example, the canonical segment *PARAGRAPH* may be indicated by *\paragraph* in one document and *\para* in another. Third, the document is scanned and inverted. Finally, the inverted text is inserted into the textbase.

*Flange*
FLANGE is a two-way command language that was developed as part of the MICROARRAS system. Consequently, it serves two major functions: it provides communication between the user interface and the analytic engine that performs all search and analysis operation, and it provides a formal specification for the system. It is written in a BNF-like notation. Consequently, programs can easily construct command expressions which, in turn, can easily be parsed. Additionally, the components of a FLANGE "sentence" are strongly typed to simplify processing and to ensure reliable transmission across a communication interface.

It is FLANGE's capability of providing a formal high-level text analysis language and its capability of delivering its results in a structured and typed form - rather than as a stream of data - that makes it feasible for an expert system to work iteratively with the textbase.

One particularly useful feature of FLANGE is its two-way communication capabilities. The following example shows a typical interaction between MICROARRAS's user interface program and its analytic engine. Suppose the user wishes MICROARRAS to display concordance information for a particular word in a text in the textbase. The user's

request for a concordance is first translated by the interface program into a FLANGE expression. That expression is then sent to the MICROARRAS engine, either running on the same machine or on a remote computer. The engine parses the message and performs the operation requested. It then encodes the results in the conventions of the return portion of FLANGE and sends that message to the user interface. The user interface parses the messages, interprets the result, and either displays the requested information to the user or engages the engine in a further FLANGE dialogue.

## 2.3 Textbase

The textbase contains the Fall, 1986 draft of *Computer Architecture, Volume 1 - Design Decisions* by Blaauw and Brooks. The manuscript consists of 188,278 words comprising 8 chapters, titled: "Introduction", "Machine Language", "Addresses", "Data", "Operations", "Instruction Sequence", "Supervision", and "Input/Output". It was chosen as the test text because

1) it was available online.
2) I was familiar with the text's content domain.
3) I was familiar with the text itself.
4) there was a ready supply of experimental subjects (fellow computer science graduate students) who were familiar with the text's content domain.
5) the text was large enough for a realistic demonstration of feasibility; there were over 3,000 paragraphs in the text providing many different possible retrieval sets.
6) the text was small enough to economically process and store online.
7) the scope of the text was broad enough to answer any of a large collection of queries.
8) the scope of the text was dense enough to provide answers to most questions within the text's coverage.

Texts to be used as MICROARRAS textbases require format marks of interest to users to be inserted in the text. TeX format marks were already present and were used as the basis for the MICROARRAS segments. These included format marks to be used in the display of the retrieved text (line, italics, label), as well as those which provide context information (chapter, section, subsection, subsubsection, paragraph, sentence, item). A series of programs are then run on the formatted text to produce an inverted file. Finally, this inverted file is converted to fixed length records for fast access.

## 2.4 Thesaurus

All domain-specific knowledge is contained in a hierarchical thesaurus. The expert system uses this information to reformulate queries. The thesaurus was built by the author from the Brooks and Blaauw text, and it strongly reflects the word usage of that textbase. For a commercial database of many texts or documents, the thesaurus would need to be broader in scope. Ideally, individual users should be able to tailor the thesaurus to better reflect their areas of interests. For example, they may wish to add specific words to the thesaurus that are too narrow in scope, or too infrequent, to be included in a general thesaurus.

### 2.4.1 Logical Structure

This section describes the structure of the thesaurus. There are several thesaurus constructs that require definition. Word types which share a common stem are grouped into *stemgroups*. The members of a given stemgroup are called *stemwords*. Each word type in the Blaauw and Brooks text appears in exactly one stemgroup. Thesaurus classes contain stemgroups which are synonyms for each other. Stemgroups may appear in zero,

one, or more than one thesaurus class. Because the thesaurus classes are linked together with parent-child links, they are also referred to as *nodes*. The arrangement of the words into stemgroups, stemgroups to thesaurus classes, and the classes into a hierarchy is discussed. Throughout this discussion, word types will be written in lowercase, stemgroup names with a leading uppercase letter, and thesaurus class names in uppercase. At the lowest level, words with the same root are grouped into stemgroups. A stemgroup contains all the words which lexically share the same root. Most are easily identified by sorting the dictionary of word types in the database. Common forms of word types not used in the textbase - for example if there was no plural of a noun - are added to the stemgroup. Consider the grouping of words with the root, *structure*.

Stemgroup Name: Structure
Stemwords: structure, structuring, structured, structures

In addition to words which were lexically similar, words which are semantically forms of the same stem were included. Thus, *run* is in the same stemgroup as *ran*. Finally, each stemgroup also contains words formed from the stem by the use prefixes. Thus, *undecided* is in the same stemgroup as *decided*.

Next, stemgroups pertaining to technical concepts are identified. Synonyms among these stemgroups are combined to form thesaurus classes. Non-technical terms are not included in the thesaurus. Extremely low-frequency stemgroups are also excluded. Low-frequency stemgroups represent unimportant, little-discussed concepts, and excluding them detracts very little from recall, while decreasing the size and complexity of the thesaurus [Salton & McGill, 1983].

High-frequency stemgroups represent broad concepts discussed throughout the text. To improve precision, most thesauri replace high-frequency stemgroups with multiple word phrases, or they exclude them altogether. In this system, I combine high-frequency technical stemgroups, for example *data* and *structure*, to form lower-frequency multiple word phrases, e.g. *data structure*, which are included in the thesaurus. However, I also include the high-frequency stemgroups, because the user is likely to use these words, and I want the expert system to be able to use the thesaurus to direct them to narrower terms. The expert system filters out these stemgroups if they are introduced during thesaurus expansion.

Finally, an ordering is imposed on the thesaurus classes. Conceptually, a thesaurus class can be viewed as a node in a lattice structure (see Figure 2.2). Each node contains a name, a list of synonym stemgroups, the names of zero or more parent nodes, and the names of zero or more child nodes. Parent nodes - nodes higher in the thesaurus structure - represent more general concepts than the current node. Child nodes - nodes lower in the thesaurus structure - represent more specific terms. Nodes containing multi-word phrases have as parents the nodes containing each of the component stemgroups. For example, consider the thesaurus entry for Data_Structure:

Node Name: DATA_STRUCTURE
Node Stemgroups: Data_Structure
Parent Node(s): DATA, STRUCTURE, NAME_SPACE
Child Nodes(s): ARRAY, QUEUE, STACK, LIST

Notice that in Figure 2.2 *vector* is shown as a synonym for *array* and *matrix*. Whereas mathematically a vector is more properly a particular type, or child, of an array, in an architectural sense they are so closely related as to be treated as synonyms. A user interested in vector processors is likely to be interested in array processors, as well.

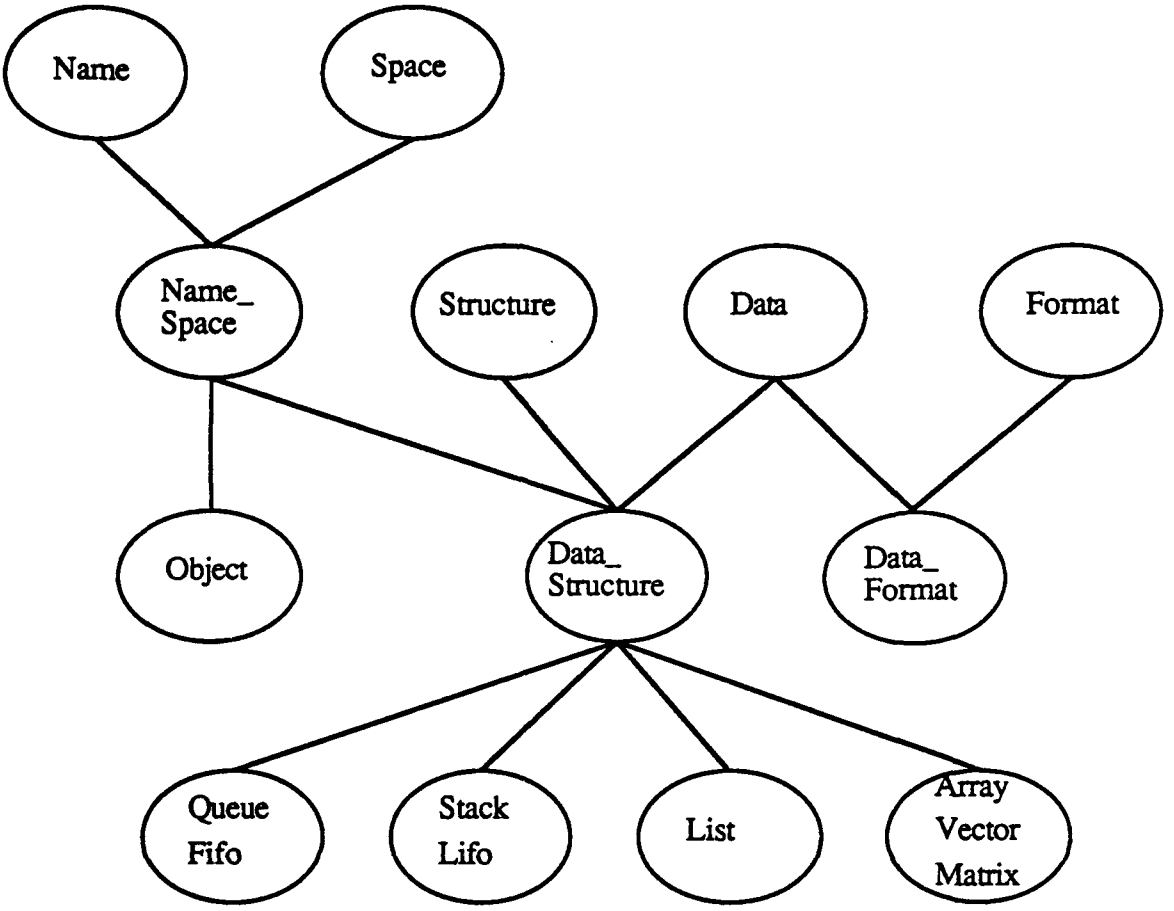



Figure 2.2. A Sample Thesaurus.

### 2.4.2 Thesaurus Words

The thesaurus was manually constructed from the 8,313 different word types in the textbase. Removing numbers, punctuation, English function words, proper names, and words which appeared only once left 5,726 types. These were grouped into 1,993 stemgroups; common word forms missing from the stemgroups were added, bringing the total to 6,990 types. Using a concordance and frequency of occurrence, 936 technical stemgroups were selected to be arranged hierarchically in the thesaurus from the 1,993 available. The construction of the thesaurus relied on the procedure outlined by Gerard Salton [Salton and McGill, 1983], the author's knowledge of computer architecture, and the hierarchical arrangement of sections in the Blaauw and Brooks text.

Before the stemgroups were placed into classes of synonyms to be arranged hierarchically, they were first partitioned into loose collections of related terms, called *buckets*. Each stemgroup was placed into one of the following buckets: architecture, sequencing, languages, hardware, formats, control operations, data operations, data representation, operating systems, memory, input/output. High-frequency stemgroups were combined to form multi-word phrases, which were added to the appropriate buckets. Ambiguous stemgroups were placed in more than one bucket, as necessary.

The synonym stemgroups in each bucket were grouped into thesaurus classes, and a lattice structure was imposed on all the classes in the bucket. The concordance was used again to

direct the formation and arrangement of the classes. Because there were few cross-bucket references, dealing with one bucket at a time allowed me to decrease the complexity of the hierarchical arrangement task . Finally, the lattices for all the buckets were merged. In all, 7,424 words in 2,253 stemgroups were arranged into 753 thesaurus classes.

### 2.4.3 ASCII Files

The ASCII files describing the contents and structure of the thesaurus are created manually. Separate files enumerate each of the words, stemgroups, thesaurus classes, and hierarchical arrangement of the thesaurus classes.

*dict_ascii* contains all the word types used in the textbase, plus the added common word forms and multi-word phrases. The words are stored in lower case, only. For the purposes of this description, *word* refers to a single word type only, whereas *dictword* refers to either a single word or a multi-word phrase. *stems_ascii* contained the grouping of dictwords into stemgroups. *nodes_ascii* contained the grouping of stemgroups into thesaurus classes. *links_ascii* contained the links connecting thesaurus classes. Each of these files will now be described in more detail.

The first file created was stems_ascii, which contains one line per stemgroup. This file is sorted in ascending order by Stemname. Multi-word phrases, e.g. data_structure, always contain an underscore.

Syntax: Stemname word+ | word_word
Examples: Data data datum
Data_Structure data_structure data_structures
Array array arrays

dict_ascii is created from stems_ascii by running a utility called *createdict.*. Each dictword in stems_ascii is placed on a separate line, followed by -1. dict_ascii is then sorted into ascending order.

Syntax: dictword -1
Examples: data -1
data_structure -1
data_structures -1
datum -1

The fixed-length version of the dictionary file includes the index in the stem file of the stemgroup containing each dictword. The ASCII dictionary file, dict_ascii, has these indices initialized to -1.

Each line of nodes_ascii defines one thesaurus class.

Syntax: NODENAME dictword+
Examples: DATA data
DATA_STRUCTURE data_structure
ARRAY array matrix vector

Whereas logically each NODENAME is a grouping of stemgroups, I found it simpler to represent a stemgroup by one of its member dictwords than to use the Stemname. Since a dictword may only appear in one stemgroup, there is no ambiguity introduced by using dictwords, instead.

The arrangement of the classes into a lattice is described in links_ascii. There is one line for each node defined in nodes_ascii. The line contains the NODENAME for the node

followed by the NODENAMES of its children, if any. Each thesaurus class in node_ascii has an entry, even if it has no children. The file is sorted in ascending order by the parent NODENAME.

Syntax: NODENAME NODENAME*
Examples: DATA DATA_STRUCTURE DATA_FORMAT
DATA_STRUCTURE ARRAY, QUEUE, STACK, LIST
STRUCTURE DATA_STRUCTURE

### 2.4.4 Binary Files

The final phase in the creation of the thesaurus was to convert the ASCII files, described in the previous section, to binary files. Most of the entities in the system (e.g. words, stemgroups, thesaurus classes) are of variable size, indicating that variable length records would provide an efficient use of space. However, I chose to use fixed length records to allow direct access to a given record via its index. Thus, I decided to trade space for time.

The thesaurus is composed of three binary files:
dict: contains a record for each dictword
stem: contains a record for each stemgroup
node: contains a record for each thesaurus class

The record structure for each of the files will be described next. The following format will be used to describe the records:

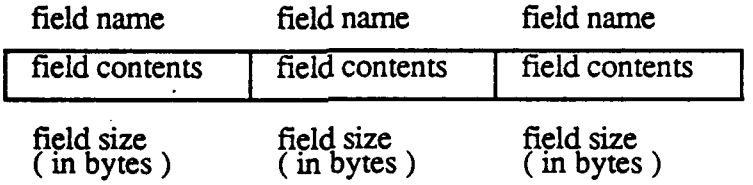



Figure 2.3. A sample record.

A dict entry has two fields, *dictword* and *dictstem* (see Figure 2.4). Dictword is MAXWORDLENGTH (the maximum dictword size) bytes long and contains a word type or multiple word phrase. Dictstem is an integer index into the stem file indicating the stemgroup to which the dictword belongs. Dict is created from dict_ascii, which contains all word types and in which dictstem is initialized to -1.

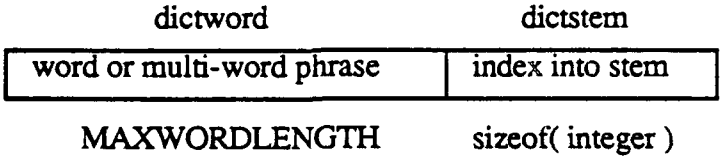



Figure 2.4. A dict record.

A stem entry has three fields: *stemname*, *stemwords*, and *stemnodes* (see Figure 2.5). Stemname is MAXWORDLENGTH bytes long and contains the name of the stemgroup. Stemname is taken directly from stems_ascii. Stemwords is an array of MAXWORDSPERSTEM (the maximum number of words in a stemgroup) integer indices into the dict file, indicating the words contained in the stemgroup. All elements of stemwords are initialized to -1, then dict is searched for each word in the stems_ascii entry, and its index is used to set the stemwords elements. At the same time, the dictstem entry

for the word is updated to contain the index of the current stemgroup. Stemnodes is an array of MAXNODESPERSTEM (the maximum number of nodes in which a stemgroup may appear) integer indices into the node file, indicating the nodes to which the stemgroup belongs. All elements of stemnodes are initialized to -1.

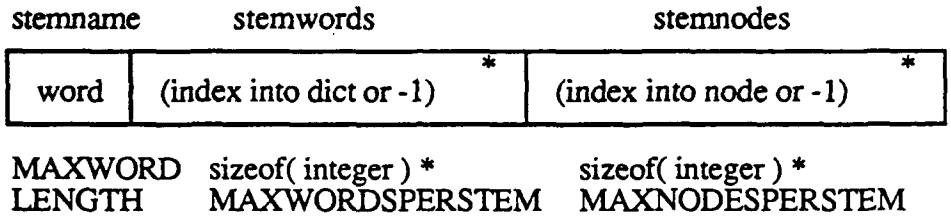



Figure 2.5. A stem record.

A node entry has four fields: *nodename*, *nodestems*, *parents*, and *children* (see Figure 2.6). Nodename is MAXWORDLENGTH bytes long and contains the name of the nodegroup. Nodename is taken directly from nodes_ascii. Nodestems is an array of MAXSTEMSPERNODE (the maximum number of stemgroups a node may contain) integer indices into the stem file, indicating the stemgroups contained in the node. All elements of nodestems are initialized to -1, then dict is searched for each word in the nodes_ascii entry, and its dictstem entry is used to set the nodestems elements. Parents is an array of MAXPARENTSPERNODE (the maximum number of parents a node may have) integer indices into the node file, indicating the parent nodes of the current node. The elements of parents are initialized to -1. Children is an array of MAXCHILDRENPERNODE (the maximum number of children a node may have) integer indices into the node file, indicating the child nodes of the current node. The elements of children are initialized to -1.

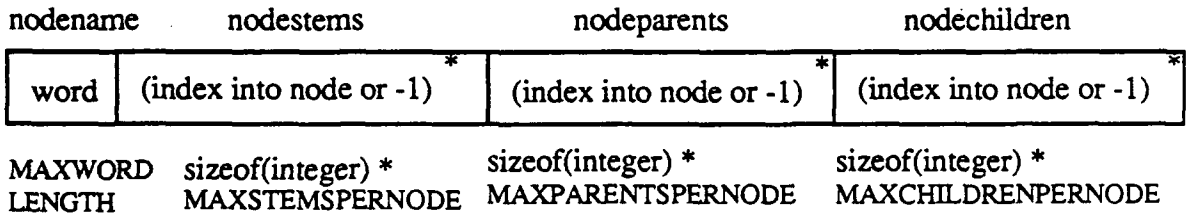



Figure 2.6. A node record.

Finally, links_ascii is processed to fill in the children and parents fields of node. As each entry of links_ascii is read, node is searched for the first nodename of the entry. The index of this nodename is the parent index. Next, the node is searched for each of the child nodenames, in turn. For each child index, the children field of the parent index entry is updated to include the child index. Also, the parent field for the child index entry is updated to include the parent index. The fixed length files now contain all the information provided by the ASCII files. Figure 2.7 shows the files created for the thesaurus shown in Figure 2.2.

### 2.4.5 Thesaurus Access Routines

The previous section outlined how the thesaurus stores information. This section discusses how that information may be accessed. First, the primitive routines which merely return information stored in the thesaurus will be described, followed by the higher level routines which implement the functions needed by the expert system. The following syntax will be used to describe the routines:

FunctionName( input parameter ) -> returned value [ | returned value ]*.

The parameter types are fields of the records described in Section 2.4.4.

| | dictword | dictstem |
|---|---|---|
| 0 | array | 0 |
| 1 | arrays | 0 |
| 2 | data | 1 |
| 3 | data_format | 2 |
| 4 | data_structure | 3 |
| 5 | datum | 1 |
| 6 | fifo | 4 |
| 7 | format | 5 |
| 8 | formats | 5 |
| 9 | formatted | 5 |
| 10 | formatting | 5 |
| 11 | lifo | 6 |
| 12 | list | 7 |
| 13 | lists | 7 |
| 14 | matrices | 8 |
| 15 | matrix | 8 |
| 16 | name | 9 |
| 17 | named | 9 |
| 18 | names | 9 |
| 19 | naming | 9 |
| 20 | name_space | 10 |
| 21 | object | 11 |
| 22 | objects | 11 |
| 23 | queue | 12 |
| 24 | queued | 12 |
| 25 | queues | 12 |
| 26 | queuing | 12 |
| 27 | space | 13 |
| 28 | spaces | 13 |
| 29 | stack | 14 |
| 30 | stacks | 14 |
| 31 | structure | 15 |
| 32 | structured | 15 |
| 33 | structures | 15 |
| 34 | structuring | 15 |
| 35 | vector | 16 |
| 36 | vectors | 16 |

(a) dict

| | stemname | stemwords | stemnodes |
|---|---|---|---|
| 0 | Array | 0 1 -1 ..... -1 | 0 -1 ... -1 |
| 1 | Data | 2 5 -1 ..... -1 | 1 -1 ... -1 |
| 2 | Data_Format | 3 -1 ..... -1 | 2 -1 ... -1 |
| 3 | Data_Structure | 4 -1 ..... -1 | 3 -1 ... -1 |
| 4 | Fifo | 6 -1 ..... -1 | 9 -1 ... -1 |
| 5 | Format | 7 8 9 10 -1 ..... -1 | 4 -1 ... -1 |
| 6 | Lifo | 11 -1 .....-1 | 11 -1 ..-1 |
| 7 | List | 12 13 -1 ..... -1 | 5 -1 ... -1 |
| 8 | Matrix | 14 15 -1 ..... -1 | 0 -1 ... -1 |
| 9 | Name | 16 17 18 19 -1.. -1 | 6 -1 ... -1 |
| 10 | Name_Space | 20 -1 ..... -1 | 7 -1 ... -1 |
| 11 | Object | 21 22 -1 ..... -1 | 8 -1 ... -1 |
| 12 | Queue | 23 24 25 26 -1.. -1 | 9 -1 ... -1 |
| 13 | Space | 27 28 -1 ..... -1 | 10 -1 .. -1 |
| 14 | Stack | 29 30 -1 ..... -1 | 11 -1 .. -1 |
| 15 | Structure | 31 32 33 34 -1 ...-1 | 12 -1 .. -1 |
| 16 | Vector | 35 36 -1 ..... -1 | 0 -1 ... -1 |

(b) stem

| | nodename | nodestems | nodeparents | nodechildren |
|---|---|---|---|---|
| 0 | ARRAY | 0 8 16 -1..-1 | 3 -1 ..... -1 | -1 ..... -1 |
| 1 | DATA | 1 -1 ..... -1 | -1 ..... -1 | 2 3 -1 ..... -1 |
| 2 | DATA_FORMAT | 2 -1 ..... -1 | 1 4 -1 .. -1 | -1 ..... -1 |
| 3 | DATA_STRUCTURE | 3 -1 ..... -1 | 1 12 -1. -1 | 9 11 0 5 -1..-1 |
| 4 | FORMAT | 5 -1 ..... -1 | -1 ..... -1 | 2 -1 ..... -1 |
| 5 | LIST | 7 -1 ..... -1 | 3 -1 ..... -1 | -1 ..... -1 |
| 6 | NAME | 9 -1 .....-1 | -1 ..... -1 | 7 -1 ..... -1 |
| 7 | NAME_SPACE | 10 -1 ..... -1 | 6 10 -1 . -1 | 8 -1 ..... -1 |
| 8 | OBJECT | 11 -1 .....-1 | 7 -1 ..... -1 | -1 ..... -1 |
| 9 | QUEUE | 4 12 -1 ... -1 | 3 -1 ..... -1 | -1 ..... -1 |
| 10 | SPACE | 13 -1 ..... -1 | -1 ..... -1 | 7 -1 ..... -1 |
| 11 | STACK | 6 14 -1 ... -1 | 3 -1 ..... -1 | -1 ..... -1 |
| 12 | STRUCTURE | 15 -1 ..... -1 | -1 ..... -1 | 3 -1 ..... -1 |

(c) node

Figure 2.7. The Sample Thesaurus's Files.

*Primitive Routines*

The primitive routines provide access to the raw data stored in the thesaurus files. Since the thesaurus was implemented using fixed-length records, information can be accessed directly. The index into the file is multiplied by the size of the file's records to provide the offset into the file of the desired record. The examples are based on the thesaurus files shown in Figure 2.7.

DSearch( dictword ) -> dictindex | error

This routine does a binary search of dict to return the index of dictword.
Example: DSearch( matrices ) -> 14.
'matrices' is the 14th record in dict.

DGetStem( dictindex ) -> stemindex | error

This routine returns the contents of the stemindex field of the dictindexth record of dict.
Example: DGetStem( 14 ) -> 8.
The 14th entry of dict (matrices) is a member of the 8th stemgroup (Matrix).

DGetWord( dictindex ) -> dictword | error

This routine returns the contents of the dictword field of the dictindexth record of dict.
Example: DGetWord( 14 ) -> matrices.
The 14th entry of dict is *matrices*.

SGetName( stemindex ) -> stemname | error

This routine returns the contents of the stemname field of the stemindexth record of stem.
Example: SGetName( 8 ) -> Matrix.
The name of the 8th stemgroup is *Matrix*.

SGetWords( stemindex ) -> stemwords | error

This routine returns the contents of the stemwords field of the stemindexth record of stem.
Example: SGetWords( 8 ) -> 14, 15, -1, ..., -1.
The 8th stemgroup (Matrix) contains dictwords 14 (matrices) and 15 (matrix).

SGetNodes( stemindex ) -> stemnodes | error

This routine returns the contents of the stemnodes field of the stemindexth record of stem.
Example: SGetNodes( 8 ) -> 0, -1, ..., -1.
The 8th stemgroup (Matrix) is a member of node 0 (ARRAY).

NGetName( nodeindex ) -> nodename | error

This routine returns the contents of the nodename field of the nodeindexth record of node.
Example: NGetName( 0 ) -> ARRAY.
The 0th node has the name *ARRAY*.

NGetStems( nodeindex ) -> nodestems | error

This routine returns the contents of the nodestems field of the nodeindex$^{th}$ record of node.

Example: NGetStems( 0 ) -> 7, 8, 16, -1, ..., -1.

The 0$^{th}$ node (ARRAY) contains the stemgroups 7 (Array), 8 (Matrix), and 16 (Vector).

NGetParents( nodeindex ) -> parents | error

This routine returns the contents of the nodeparents field of the nodeindex$^{th}$ record of node.

Example 1: NGetParents( 0 ) -> 3, -1, ..., -1.

The 0$^{th}$ node (ARRAY) has node 3 (DATA_STRUCTURE), as its parent.

Example 2: NGetParents( 3 ) -> 7, 1, 12, -1, ..., -1.

The 3$^{rd}$ node (DATA_STRUCTURE) has nodes 7 (NAME_SPACE), 1 (DATA), and 12 (STRUCTURE) as its parents.

NGetChildren( nodeindex ) -> children | error

This routine returns the contents of the nodechildren field of the nodeindexth record of node.

Example: NGetChildren( 3 ) -> 9, 11, 0, 5, -1, ..., -1.

The 3$^{rd}$ node (DATA_STRUCTURE) has nodes 7 (QUEUE), 11 (STACK), 0 (ARRAY), and 5 (LIST) as its children.

*Higher Level Routines*

The primitive routines described above can be used to implement the thesaurus functions needed by the expert system. The following higher level thesaurus access routines are used by the expert system:

1) map from a word to its stemgroup.
   The algorithm, in pseudo-C, is:
   DSearch( word ) -> dictindex
   DGetStem( dictindex ) -> stemindex

2) map from a stemgroup to the set of all words with the same stem.
   The algorithm, in pseudo-C, is:
   SGetWords( stemindex ) -> stemwords
   For each stemword
       DGetWord( stemword ) -> dictword

3) map from a stemgroup to the set of all nodes containing that stemgroup. Once the nodes are found, their names can be displayed.
   The algorithm is:
   SGetNodes( stemindex ) -> stemnodes
   For each stemnode
       NGetName( stemnode ) -> nodename

4) map from a stemgroup to the set of all synonym words for that stemgroup.
The algorithm is:
SGetNodes( stemindex ) -> stemnodes
For each stemnode
    NGetStems( stemnode ) -> stemindex
    SGetWords( stemindex ) -> stemwords
    For each stemword
        DGetWord( stemword ) -> dictword

5) map from a stemgroup to the set of more general (parent) words.
The algorithm is:
SGetNodes( stemindex ) -> stemnodes
For each stemnode
    NGetParents( stemnode ) -> nodeparents
    For each nodeparent
        NGetStems( nodeparent ) -> nodestems
        Map from nodestems -> words as in 1.

If the stemnode is the root of the tree, i.e. has no parents, all the node indices in nodeparents will be -1.

6) map from a stemgroup to thesaurus classes of more specific words. This would be done in a manner similar to 4, substituting NGetChildren for NGetParents. If the stemnode is a leaf of the tree, i.e. has no children, all the node indices in nodechildren will be -1.

7) map from a stemgroup to thesaurus classes of related (sibling) terms.
The algorithm is:
Map from stemgroup -> parentnodes as in 5.
For each parentnode
    Map from the parentnode -> childnodes as in 6.
    For each childnode
        Map from the childnode -> nodestems.
        Map from nodestems -> words as in 1.

8) determine the hierarchical relationship, if any, between two words. This can be done by mapping from each of the words to their node sets and then looking for relationships between the nodes. In this implementation, the routine looks for self, parent, child, or sibling. In all other cases, it returns none.

Other functions could be defined from the primitives, if needed. For example, the next most useful functions might be grandparent, grandchild, cousin, etc.

### 2.5 User Interface

This section presents the current user interface for the research prototype. A basic command line interface was used. The interface implemented for this research prototype is straightforward and not one of the major thrusts of the project.

#### 2.5.1 Query Language

When the user starts the system, the following prompt appears:
    Enter a query, or quit, terminated by <return>:
The system expects a Boolean query. A Boolean query language was chosen because it is

the most common type available on existing systems. I wanted the main difference between this prototype and conventional full-text retrieval systems to be the searching knowledge base so that any improvement in search performance could be attributed to the encoded search strategies, rather than the user interface. Possible improvements to the interface are discussed in Section 5.1.1.

*Operators*
The operators provided, in decreasing order of operator precedence, are: ANDNOT, AND, and OR. A logical equivalent to any Boolean expression can be constructed using these operators. Where there are two or more operators of equal precedence, they are evaluated left to right. Parantheses have the highest priority and can be used to override the default order of evaluation. The operators are distinguished from the search words by their position. A formal grammar for the query language appears later in this section.

*Search Terms*
When a query is parsed, the expert system interprets each search term to represent a unique area of interest, or *concept*, specified by the user. The concepts, and the operators, are flagged as positive or negative based on whether they are specifying information the user does, or does not, wish to receive. For example, the query 'i/o ANDNOT (device OR interrupt)' contains three concepts: *i/o*, *device*, and *interrupt*. *I/O* is a concept on which the user wishes information, so it is considered a positive concept. *Device* and interrupt indicate concepts on which the user does not wish information, so they are considered negative concepts. The ANDNOT and OR operators are followed by negative concepts, so they too are flagged as negative.

*Context*
A default context of one sentence is used for the AND and ANDNOT operators. For example, 'virtual AND memory' will retrieve all passages in which *virtual* and *memory* appear within the same sentence, regardless of order. Similarly, 'page ANDNOT fault' will retrieve passages in which *page* appears, but not those in which it appears within the same sentence as *fault*.

When the user is searching with the expert system, the expert system controls the context. Initially, the default of one sentence is used, but the expert system may adjust the context during query reformulation. However, when the user is searching without the expert system, the AND and ANDNOT operators may be augmented with a user-specified context. The user may define the search context for AND or ANDNOT in terms of words, sentences, or paragraphs. The most general context definitions have the form:

left-expression operator [integer1 to integer2 units] right-expression

where integer1 must be smaller than or equal to integer2, and units is either words, or sentences, or paragraphs. The integers specify the range around the tokens satisfying the left-expression (tokenL) in which the tokens satisfying the right-expression (tokenR) must appear. 0 represents the unit containing the token from the left hand side. Thus, the default context of one sentence is equivalent to

0 to 0 sentences

indicating that tokenR must appear in the same sentence as tokenL. An abbreviation for this context, *sentence*, is provided for the user's convenience.

Negative integers indicate that tokenR must precede tokenL; positive integers indicate that it must follow. Thus,

-5 to +3 words

specifies that tokenR must appear in the region around tokenL that includes the 5 words preceding tokenL and the 3 words following it. If one is looking for paragraphs containing

a specific phrase, for example *computer architecture*, one would use the query

computer AND [+1 to +1 words] architecture

which requires that *architecture* immediately follow *computer*. There is also a shortform for this relationship, called *nextword*.

Finally, since the retrieved passages are one paragraph long, search expression contexts should not be larger than one paragraph. Thus, the only valid context involving the unit paragraph is

0 to 0 paragraphs

and this context is abbreviated by *paragraph*.

When user-defined contexts are evaluated, they may cross paragraph boundaries. For example, if the user has specified a context of plus or minus three sentences, tokenR may appear in a different paragraph than tokenL. In this case, MICROARRAS retrieves the paragraph containing tokenL. In contrast, when the expert system controls the context, it builds more complicated contexts which eliminate hits if the tokens do not appear in the same paragraph.

*Formal Grammar*

The context-free grammar for the query language described above is specified in modified Backus-Naur form. Syntactic constructs are denoted by capitalized English words, literal tokens appear in boldface, and optional constructs appear between meta-brackets { and }. Constructs superscripted by + may appear one or more times.

Expression -> AndTerm | Expression Or AndTerm
AndTerm -> AndnotTerm | AndTerm And { [ Context ] } AndNotTerm
AndNotTerm -> Factor | AndNotTerm Andnot Factor
Factor -> Word | ( Expression )
Or -> **or** | **OR**
And -> **and** | **AND**
Andnot -> **andnot** | **ANDNOT**
Context -> Integer To Integer Unit | **nextword** | **sentence** | **paragraph**
Unit -> **word** | **sentence** | **paragraph** | **words** | **sentences** | **paragraphs**
To -> **to** | **TO**
Integer -> { + | - } Digit+
Digit -> **0** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9**
Word -> **printable character**+ Terminator
Terminator -> **tab** | **newline** | **space** | **[** | **]** | **(** | **)**

The above grammar contains left-recursion, which must be removed to allow a recursive descent parser to be built for the language. The definitions for Expression, AndTerm, and AndnotTerm are rewritten as follows (the Greek symbol epsilon (ε) represents the empty string):

Expression -> AndTerm Expression'
Expression' -> Or AndTerm Expression' | ε
AndTerm -> AndnotTerm AndTerm'
AndTerm' -> And { [ Context ] } AndnotTerm AndTerm' | ε
AndnotTerm -> Factor AndnotTerm'
AndnotTerm' -> Andnot { [ Context ] } Factor AndnotTerm' | ε

### 2.5.2 Guidance Requests

The expert system can run independently or interact with the user during query reformulation. When the system is run interactively, the expert system makes two types of guidance requests: after each reformulation step, if the reformulation is not complete, it asks the user whether or not he wishes to continue query reformulation; and when search terms (other than stemwords) are selected from the thesaurus, the user is asked whether or not to add them to the appropriate concept group. The prompts for the two requests are:

<Query is shown here.>
Passages retrieved by the previous query: *n*. Decision: broaden or narrow or success.
Continue reformulation? Enter y or n <return>:

Add stemgroup *stemname* to concept *original word*? y or n <return>:

### 2.5.3 Passage Presentation

Once the system has completed query reformulation, either because it reached the target number or the user stopped the reformulation process, the user is asked whether or not he wishes to read the retrieved passages with the following prompt:

Query n retrieved m passages.
What next? d (display) or q (new query) <return>:

If he asks to view the passages, he will see the a 'Retrieving text.' message if there are fewer than MAXPASSAGES in the retrieval set. MAXPASSAGES is set to 25 for this version of the prototype. For the experiment, after each passage the user was asked for relevance feedback with the following prompt:

Enter 0 (not relevant), or 1 (somewhat relevant), or 2 (very relevant) <return>:

If there are more than MAXPASSAGES in the retrieval set, the message 'Too many passages to display' is generated and the user is prompted to enter a new query.

## 2.6 Expert System

This section contains an overview of the expert system, a description of its interactions with other system components, and a sample scenario. The knowledge base contents which cause the actions described here are discussed in Chapter 3.

### 2.6.1 Overview

The expert system performs three main functions: it acts as system controller, it reformulates the Boolean query based on previous search results, and it ranks the retrieved passages in decreasing order of estimated relevance for presentation to the user. To perform these functions, it uses a knowledge base of search strategies and text analysis procedures. As pointed out earlier, all domain knowledge is contained in the thesaurus.

The expert system interacts with three other system components (see Figure 2.1): the user interface, MICROARRAS, and the thesaurus. This section describes the expert system's interactions with MICROARRAS. The expert system's use of the thesaurus is described in Section 2.4.5, and the user interface interactions are described in Section 2.5.2.

### 2.6.2 Interaction with MICROARRAS

*System Startup*
The system as a whole is started by invoking the expert system. This causes OPS83 to start and to load the expert system rules and C code. An OPS83 prompt is displayed and

the working memory element start is created by entering *+start*. Entering *xx* to the next OPS83 prompt begins the rule matching, which is the heart of the expert system. The execution ends when no rules can be matched with the contents of working memory. A final OPS83 prompt is displayed, and entering *q* exits the system.

The expert system can run on the same workstation as MICROARRAS, or it can run the retrieval engine on a remote workstation. In the first case, the expert system forks off a child process which runs MICROARRAS. The two processes communicate through pipes opened by the expert system. In the second case, the expert system creates a process on a remote workstation using rexec to execute MICROARRAS and the two communicate over the network.

*FLANGE*
The expert system sends and receives FLANGE messages to interact with MICROARRAS. The messages sent and received are placed in a 1025 byte buffer (1024 bytes plus 1 byte for a null terminator). If a reply from MICROARRAS is too long to fit in the buffer, this condition is flagged by placing a continue character at the end of the message. This flag causes the buffer to be emptied and a continue message to be sent to MICROARRAS to refill the buffer. There are several C routines written to pack and unpack the various FLANGE messages, and to synchronize the buffer access.

The expert system uses many of MICROARRAS's capabilities. Context specification is used to define passages of length one paragraph and default searching contexts of one sentence and, in the case of multi-word phrases, one word. Categories are created to contain the search terms (one per stemgroup or multi-word phrase). Frequency information in the textbase as a whole is requested for each word as it is entered as a search term. During passage ranking, frequency information is requested for each search term in each retrieved passage. Finally, the retrieved passages are requested for display to the user.

### 2.6.3 Sample Scenario

Now that the system components have been described individually, a sample scenario will be presented to illustrate how they work together to provide an intelligent online search assistant. Since our current textbase concerns the domain of computer architecture, the following example describes the interactions of the system and a user searching for information on the alignment of word boundaries in memory. The contents of the knowledge base which cause the following query reformulations will be describe in detail in chapter 3.

The user might enter a query 'boundary AND word ANDNOT page', which indicates that he wishes to retrieve passages containing information on word boundaries but not page boundaries. Assume a target number of 15. Applied to this textbase, the original query would retrieve only one passage, so the expert system would attempt to broaden the query. The first step would be to replace the word types *boundary* and *word* with their stemgroups. The resulting query would be 'Boundary AND Word ANDNOT page', where the capitalized search terms indicate the whole stemgroup is included. Notice that *page* has not been expanded to its stemgroup, as it is a negative, or excluded, concept. Four passages would now be retrieved.

The next step would be to broaden the query by including synonym stemgroups for each of the positive search terms, in turn. From the thesaurus it is found that Boundary has one synonym, Limit, however there is no synonym for Word. The query now becomes

'(Boundary OR Limit) AND Word ANDNOT page', which retrieves seven passages. Relaxing the context around the AND operator to adjacent sentences while decreasing the context around the ANDNOT operator to within 5 words increases the number of passages retrieved to nine. To further broaden the query, the parent stemgroups for the positive concepts are added. Block and Segment are added to the concept Boundary. The Word concept remains unchanged, since Word has no parent in the thesaurus. The query becomes '(Boundary OR Limit OR Block OR Segment) AND Word ANDNOT page', which retrieves twelve passages. Twelve is within 20% of the fifteen passages requested, so the reformulation stops. If the user requests to see the retrieved passages, the expert system would rank the retrieved passages and present them to the user in decreasing rank-order. This procedure is described in Section 3.5.

If the number of passages requested had been larger, for example, 35, the expert system would have continued to broaden the query by adding more stemgroups from the thesaurus, relaxing context, and changing the query structure. First, the positive siblings would be added. Free, Frame, Fault, Fragment, and Scatter would be added to the Boundary concept, although none is particularly useful for this query. Page is a sibling of Boundary, but it would not be added because *page* already appears in the query. The sibling stemgroups Align, Bit, and Byte would be added to the Word concept, increasing the number of passages retrieved to nineteen.

Broadening would be continued by adding the child stemgroups. Align is a child of Boundary, but it would not be added to that concept because it already appears in the Word concept. Word_Size and Word_Mark are added to the Word concept, but they have no effect because they are multi-word phrases containing *word*, and this concept already contains *word* on its own. Next, the context required between the stemgroups would be relaxed to the maximum (same paragraph for positive concepts, adjacent words for negative concepts). This would increase the number of passages retrieved to 26. Dropping the negative part of the query, 'ANDNOT page', would be tried next, but it would have no effect on the number of passages retrieved. Finally, the AND would be changed to OR. This substitution would retrieve far too many passages, resulting in over 100 passages, so the system would backtrack to the previous query.

The final query, '(Boundary OR Limit OR Block OR Segment OR Fault OR Fragment OR Scatter) AND (Word OR Align OR Byte OR Bit OR Word_Size OR Word_Mark)', would retrieve 26 passages. The user could choose to display these or to restart the system with a different query in an attempt to retrieve more information.

# *Chapter 3*
# *Searching Knowledge Base*

## 3.1 Overview

### 3.1.1 Knowledge Base Contents

Professional search intermediaries use three main types of knowledge - their knowledge of how particular databases are constructed, knowledge about the domain being searched, and knowledge of general search strategies - to form and improve queries. The expert system handles all interactions with MICROARRAS, the text retrieval software used; the user will need no specific knowledge of this system. Domain knowledge is all incorporated in the hierarchically structured thesaurus. This chapter discusses the knowledge base of search strategies that forms the core of the expert system.

The expert system performs three main functions:

1) it controls the operation of the system as a whole;
2) it reformulates the Boolean query based on previous search results;
3) it ranks the retrieved passages in decreasing order of estimated relevance for presentation to the user.

To perform these functions the expert system contains a knowledge base of the search process, search strategies, and passage ranking procedures.

### 3.1.2 Knowledge Base Design

In conventional programs, design is discussed in terms of data structures and algorithms. The analogous discussion for an expert system covers working memory elements, the knowledge base, and the inference algorithm. The working memory elements contain the data available to the expert system describing the current state of affairs, whereas the knowledge base is a collection of condition-action pairs, called *rules*, which execute based on the current contents of working memory. The inference engine controls the choice of which rule executes, or *fires*, if more than one has its conditions satisfied.

If more than one rule has its conditions satisfied, the conflict resolution strategy used in this system will first choose the rule or rules with the largest number of conditions. These rules are chosen because they handle more specific cases than rules with fewer conditions. If there is still more than one candidate rule, the rule matched by the most recently created working memory elements is chosen, leading to a type of depth-first execution.

Knowing the inference algorithm, the system execution can be explained in terms of the rules in the knowledge base and the working memory elements that cause them to fire. Section 3.2 describes the working memory elements, whereas sections 3.3, 3.4, and 3.5 describe the rule bases for system control, query reformulation, and passage ranking, respectively. The complete rule base appears in Appendix I'. This knowledge base is drawn from the published literature on searching (see Section 1.3.5), search training manuals, consultation with professional searchers, and experience with sample searches.

## 3.2 Working Memory Elements

A description of the seven types of working memory elements, or *elements* , follows, with an indication of how many instances of each type are created:

1) start: created to startup the system; one per user session
2) goal: the current high-level goal; one per user query
3) reform: contains information about the reformulation state; one per user query
4) query: contains information about the query as a whole; one per user query
5) concept: contains information about the concept group; one per concept
6) stem: contains information about the stemgroup; one per stemgroup
7) passage: contains information about the retrieved passage; one per passage

Several attributes contain indices into C structures which duplicate the information. Since much of the reformulation work is accomplished by C routines under the direction of the expert system, indices are passed to avoid exchanging large amounts of data between the two modules. As they merely duplicate information, or store additional information merely for efficiency reasons, the C structures will not be described in this dissertation.

*start.* No attributes.

Created upon system startup, it triggers the creation of a MICROARRAS process and a goal element, then is removed.

*goal.* Attributes:

type: current goal, e.g. *get* query, or reformulate
subgoal: temporary goal during reformulation, e.g. addstemword
justright: target number entered by user
min: minimum number of passages acceptable, set to justright * 0.5
max: maximum number of passages acceptable, set to justright * 1.5
toofew: lower bound on success, set to justright * 0.8
max: upper bound on success, set to justright * 1.2

The first goal type is *get query*. This causes a rule to fire which receives a query from the user and creates the query, concept, and stem elements. Each search term entered by the user is assumed to represent a distinct concept and a concept and stem element is created for each term.

*query*. Attributes:

status: indicates query's reformulation status, e.g. new, or reformulated, or final
version: index into C array for this query's parse tree
name: name of MICROARRAS's representation of this query
numpassages: the number of passages retrieved by this query
context: number representing the current context for the query
hi: the broadest context tried with this query so far
lo: the narrowest context tried with this query so far

Only one query is active at a given time. The user's query is parsed and information about its structure is stored in a C array of parse trees. When the Boolean operators are changed or removed, the current query's status is set to *reformulated* and a new parse tree is created. A new query element is created with a new version and name. Changes to context or sets of search terms do not require new query element, or new parse tree, merely an update to the name field to reflect the new MICROARRAS category, or the context and hi or lo fields to reflect the new context. The numpassages field is also updated to reflect the number of passages retrieved at each step along the way.

*concept.* Attributes:

status: indicates whether or not this concept is in current query, e.g. active
id: index into C array for this concept
sign: indicates whether concept is positive or negative, e.g. + or -
freq: number of occurrences of all stemgroups in the concept in the textbase
state: the last reformulation performed on the concept, e.g. addsynonyms

A concept element is created for each search term in the user's initial query. As the query is reformulated, search terms may be added to the concept by the expert system. These are stored in a C structure whose identity is stored in the id field. During query reformulation, negative concepts may be dropped. In order to allow backtracking to an earlier query, the concept's working memory element is not deleted in this case, but merely has its status field set to *inactive*. The concept's sign is stored in the element to allow different processing of positive and negative concepts. A concept's frequency is set to the sum of the frequencies of all the search terms in the concept. Additionally, information about the state of the reformulation process is stored for each concept.

*stem.* Attributes:

id: index into C array for this stemgroup
concept: identifier for the concept containing this stemgroup
name: English name for the stemgroup, e.g. boundary
freq: number of occurrences of this stemgroups in the textbase
added: reformulation step which caused this stemgroup to be added to the query, e.g. 5
dist: distance of stem in thesaurus from original search term

Initially, a stem element is created for each concept in the user's query. This stem element initially contains only the user's search term. Reformulation may cause the other members of the stemgroup to be added, which updates the freq field, and adds search terms to the C structure for this stemgroup, but does not cause a new stem element to be created. Further reformulation may add whole new stemgroups to the concepts, causing new stem elements to be created. The dist field stores the *distance* of the stem from the original search term. It contains 0 for the original term, 1 for the rest of the stemwords in this stemgroup, 2 for synonyms, 3 for parents, 4 for siblings, and 5 for children.

*reform.* Attributes:

global: the type of reformulation required by the user's initial query, either broaden, or narrow
local: the opposite direction of global, e.g. if global is broaden, local is narrow
next: the direction of the next reformulation, i.e. broaden or narrow
lastglobal: the last reformulation technique tried in the global direction
lastlocal: the last reformulation technique tried in the local direction
laststate: the last reformulation technique
step: counts the number of reformulations performed
dist: distance of new search terms from original (used when creating new stem elements)

If the user's query needs reformulation, i.e. does not retrieve the target number of passages, a reform element is created. The type of reformulation to be performed on the user's query is stored in global. The sample query of 'boundary AND word ANDNOT page' would retrieve one passage. If the target number was 15, global would be set to *broaden*, local to *narrow*. Lastglobal, lastlocal, and laststate would all be set to *original* as no reformulation has yet been done. Step is initialized to 0.

Global and local remain unchanged during reformulation. Next indicates the direction of the required reformulation. Lastglobal, lastlocal, and laststate are updated to keep track of the progress of the reformulation process. Step is incremented by one for each reformulation technique tried. Dist is set by calling C reformulation routines which return the thesaurus distance information for the search terms proposed. For example, if the above query were broadened by adding stemwords, it would result in the retrieval of four passages. Because four passages are not enough to meet the user's target number, next would be set to *broaden*. Since the reformulation added search terms that were in the same stemgroup as the original search term, laststate would be set to *addstemwords*, and dist would be set to 1. Because the previous reformulation was a broadening technique, and the global direction is *broaden*, lastglobal would be set to *addstemwords*. Lastlocal would remain unchanged, and step would be incremented to 1.

*passage*. Attributes:

status: new, or ranked, or displayed
id: index into C array storing pointers to the reconstructed passages
wt: calculated weight of this passage

After the query reformulation stops, the passages corresponding to the final query are retrieved, weighted, and displayed in decreasing order of weight. A passage element is created for each passage, and information stored on it as indicated above.

## 3.3 System Control Rules

The system control rules are used to determine the execution path of the system. These rules execute based primarily on the contents of the *start* and *goal* working memory elements. As these working memory elements are created and modified, they satisfy the condition parts of different reformulation rules and cause them to fire. Figure 3.1 shows the flow of control of the system, with rules shown as nodes in the graph. The elements that cause them to fire, with their attributes in parentheses, label the arcs. As a query is reformulated, control cycles between the Reformulate and WhatNext nodes. The reformulation tactics appear in detail in Figure 3.2.

The system is started by the user creating the start element. This causes the rule Init to fire which removes start and creates the goal element with the type field set to *getquery*. GetQuery, SendQueryToMA, and CheckInitialResults then fire in sequence by modifying the type field of goal. These rules initialize the elements described in Section 3.2 and begin the reformulation process described in Section 3.4. Reformulation continues until success (retrieving between toomany and toofew passages) or failure (running out of reformulation techniques) occurs. In the interactive version of the system, reformulation can also be stopped by the user after any step. After Success or Failure fire, the user may display the passages, in which case WeightPassages and DisplayPassage fire. If there are too many passages, or too few, he may choose not to display the passages, but rather to begin with a new query. This causes a collection of RemoveWME rules to fire, removing the working memory elements, and the system restarts at GetQuery.

## 3.4 Query Reformulation Rules

### 3.4.1 Overview

Queries are reformulated based on the target number, the number of passages retrieved, and the history of broadening and narrowing techniques already applied. The expert system has a collection of reformulation tactics at its disposal. Marcia Bates [Bates, 1979] and others have identified successful search tactics. However, no one has outlined an overall

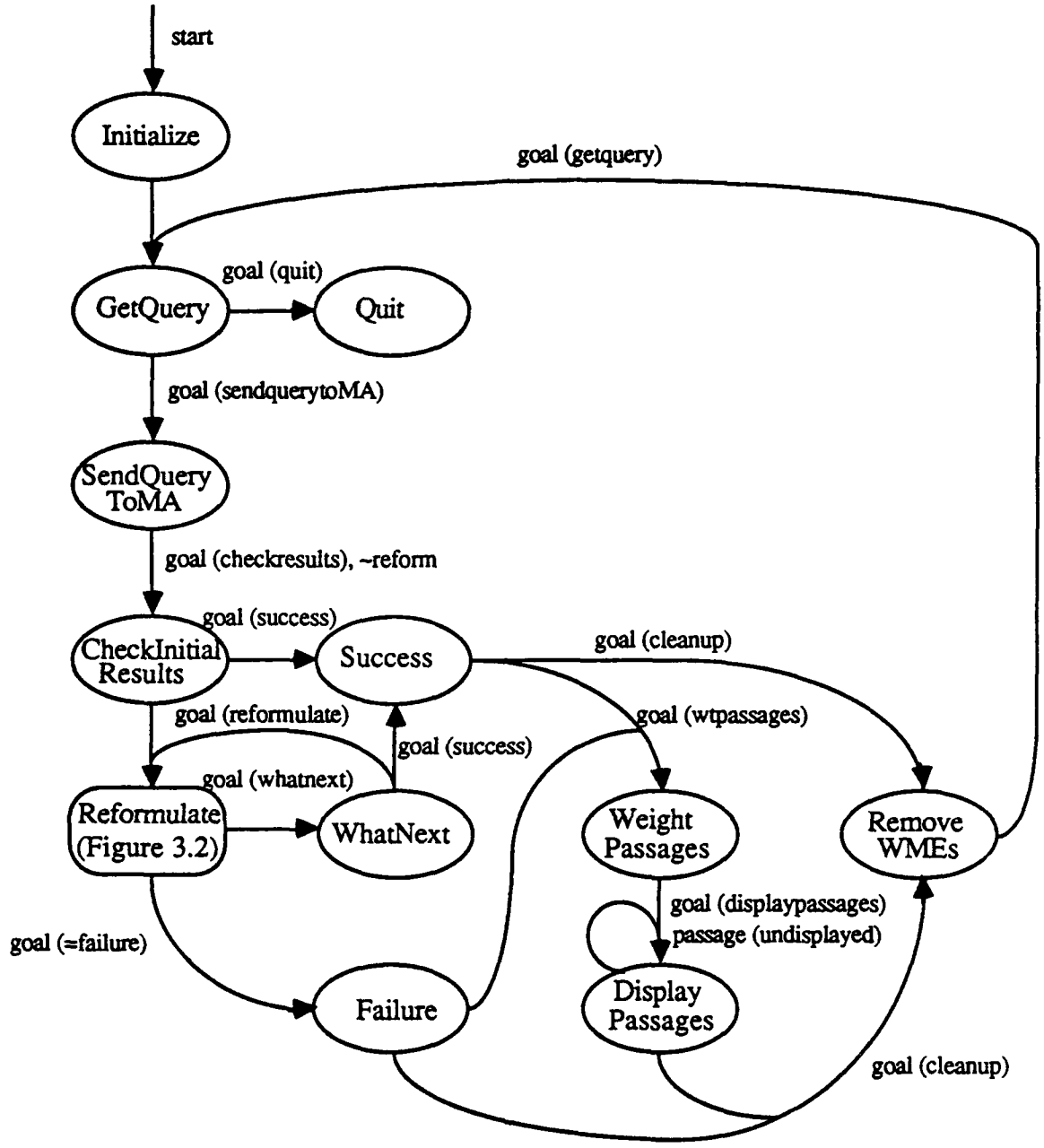



Figure 3.1 System Execution Cycle

query reformulation strategy combining these tactics. The guiding principles for the expert system's query reformulation knowledge base were: 1) each search term in the initial query represents one concept on which the user does, or explicitly does not, want information; 2) the user's initial search terms are the best indication of the user's areas of interest; 3) some terms from the thesaurus may be helpful, but others will not; 4) to broaden a concept using the thesaurus, parent stemgroups should be added first, followed by siblings, then children 5) the expert system should never discard concepts in which the user has indicated an interest.

Each search term in the initial query is expected to represent a unique concept. This needs to be explained to the users of the system. Carolyn Crouch [Crouch, 1988] found that

augmenting a query with thesaurus terms, rather than replacing the original search terms, lead to improved results. With this in mind, concepts are expanded by adding thesaural terms (ORing them with the terms already in the concept) rather than by replacing the terms already present.

The importance of the initial search terms is reflected in the query reformulation strategy by the decision to use context to broaden/narrow the query immediately after the terms have been expanded to their stemgroups. Before adding new stemgroups from the thesaurus, I felt that it was desirable to attempt to reach the target number using context alone.

The belief that some stemgroups from the thesaurus will be useful, while other will not, is the basis for providing user filtering of the candidate thesaurus terms. The domain-dependent search strategies identified by Philip Smith [Smith, 1989] involved the use of domain knowledge to choose the appropriate terms from a thesaurus. In addition, Donna Harman [Harman, 1988] showed that search results improved when thesaural terms were filtered by the user. Based on these two studies, I decided to allow the users to select which stemgroups to add from a set of thesaural candidates.

To expand a concept, he order in which the terms are added from the thesaurus is: parents, then siblings, then children. Replacing a term with its parent to broaden a query is a common practice, both by searchers [Bates, 1979; Salton, 1988], and in systems which automatically reformulate queries [Vickery and Brooks, 1987; Chiaramella and Defude, 1987]. The rationale is that since parent terms represent broader concepts, adding the parent term should broaden the scope of the query. Thus, parent terms are added first. Siblings are added second since they represent related concepts, and children terms are added third since they represent narrower concepts and seem less likely to broaden the concept. While the expert system uses this ordering, the reasoning is based on experience with searching bibliographic databases using keywords. In full-text databases, I believe that the reverse order may make more sense. Broadening a concept containing *apple* with children terms, yielding 'apple OR mcintosh OR granny_smith', seems more likely to retrieve relevant passages than broadening with the parent terms, yielding 'apple OR fruit'. This tactic needs to be explored in future versions.

Finally, the expert system never discards positive concepts. If the user has indicated more than one concept in which he is interested, the expert system has no knowledge from which to decide which concept is more important to the user. Thus, if expanding the query using the thesaurus and context still yields too little information, the expert system does not drop one of the concepts in an attempt to broaden. Rather, it ORs the concepts together to retrieve the available information on all the concepts. In contrast, negative concepts are dropped from a query if the other broadening techniques fail.

Figure 3.2 diagrams the flow of control among the reformulation techniques. The left side of the Figure 3.2 diagrams the broadening techniques, the right side the narrowing techniques. This figure is somewhat simplified since it does not show the use of context to converge to the target number once queries have been found which bracket the target number from above and below. The addition of search terms, including the handling of

multi-word phrases, is described in more detail in Section 3.4.2, context adjustment in Section 3.4.3.

*Flow of Control*

After each reformulation, there are three possible outcomes, broaden, narrow, and success. The node for each reformulation step contains an arc specifying the action taken for each of

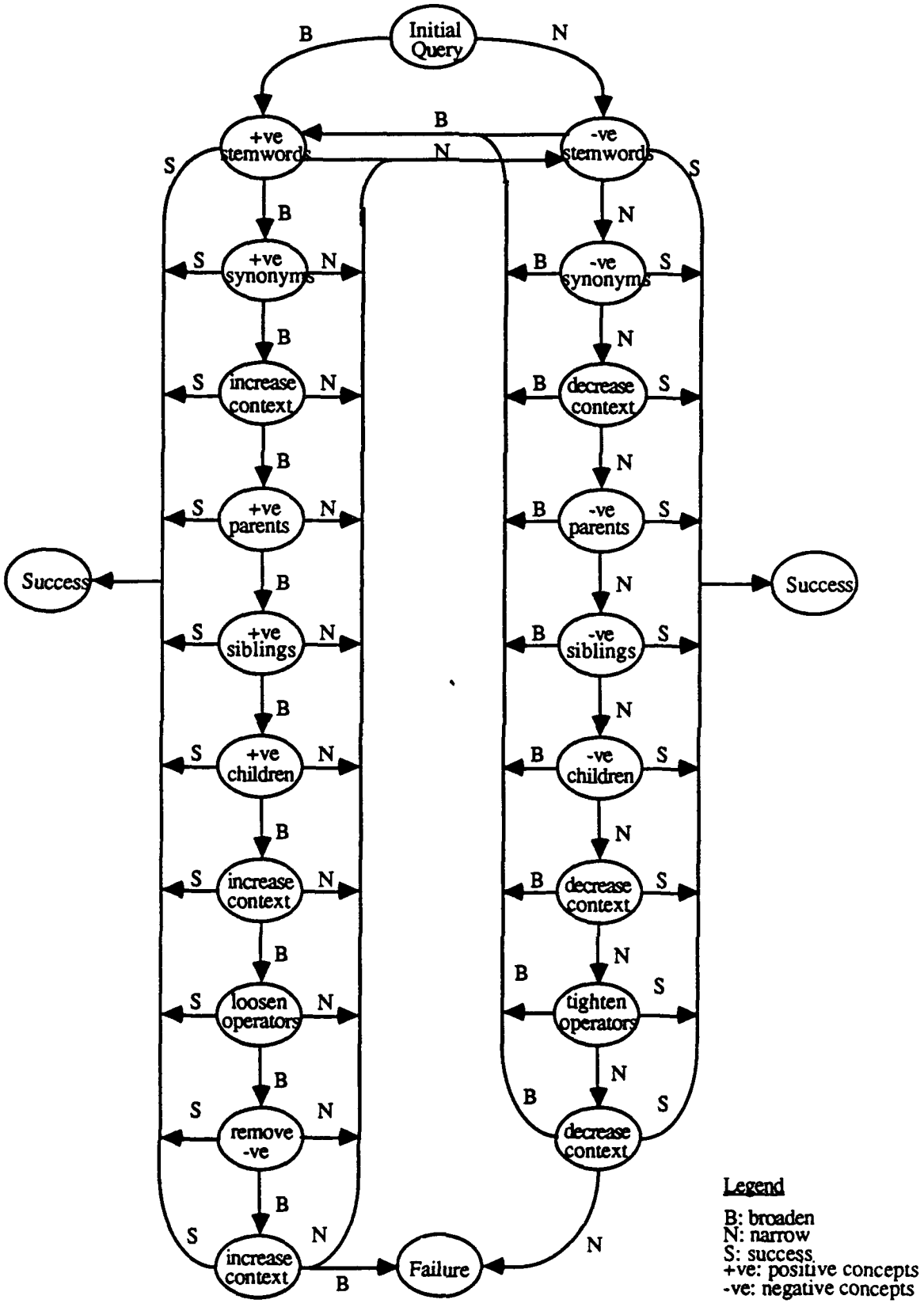



Figure 3.2 Query Reformulation Techniques

the possible results. Success at any reformulation node leads directly to the Success node. While reformulation continues in the initial global direction, the nodes down that side of the diagram are visited in turn.

The earlier techniques on both sides of the diagram represent those which are expected to be most useful. Thus, the first reformulation in the opposite, or local, direction moves the reformulation to the node at the top of the opposite side of the diagram. For example, if the expert system has broadened a query five times in a row, it would be in the Add Positive Siblings state. If too many passages are now retrieved, the expert system would attempt narrowing by adding negative stemwords, the first state at the top of the narrowing side of the diagram.

Reformulation never continues in the local direction farther than it preceded in the global direction. Queries have already been formed which bracket the target number from below and above, otherwise the system would not have tried both narrowing and broadening techniques. Rather than using techniques which are considered less likely to produce good results, such as adding more distant thesaurus terms or manipulating the query structure, the expert system adjusts the context.

Continuing the above example, if the expert system then narrowed the query five times, it would be in the Add Negative Siblings state. If the query required further narrowing, the expert system would not go to the next state down, Add Negative Children, because it did not get that far in the global broadening direction. If context adjustment does not result in success, the expert system chooses the most recent query which resulted in too many passages, rather than one which results in too few.

*Broadening*

Examining the figure in more detail, one can see that the first broadening technique used is adding stemwords to positive concepts, followed by adding synonyms. Next, the expert system increases context simultaneously in three ways: strict adjacency between terms in multi-word phrases is relaxed to the component words appearing, in any order, within three words of each other; the context around positive operators is loosened from the same sentence to +/- one sentence; and the context around negative operators is tightened to +/- seven words. Related terms from the hierarchical thesaurus are added next: words from parent classes first, followed by siblings, and finally children. Context is then further broadened such that terms from multi-word phrases are required to appear within the same sentence, positive operators are evaluated with a context of the same paragraph, and negative operators have their context decreased to +/- three words.

More drastic approaches are attempted if the previous techniques do not broaden the query enough. These affect the Boolean operators in the query. First the positive operators are loosened from AND to OR, while negative operators are tightened from OR to AND. If the query still requires broadening the expert system removes the negative portions of the query altogether.

*Narrowing*

Narrowing techniques are identical to broadening techniques but are applied to the opposite parts of the query. Narrowing is accomplished by expanding terms in the negative concept groups, tightening positive context, loosening negative context, tightening positive operators, and loosening negative operators. The right side of Figure 3.2 shows the order in which these techniques are applied.

*Stopping*
Marcia Bates [Bates, 1979] stated that knowing when to stop a search is a difficult problem. I partially side-step this problem by having the user explicitly state the number of passages he wishes to retrieve. Since the target number he supplies is likely to be a rough guess, a range of 20% is considered successful. A larger range may be desirable, but since the user is able to stop the reformulation process himself, the size of the range is not important. Left on its own, the expert system stops the reformulation process when it retrieves within 20% of the target number, or it has run out of techniques to try. The rules to implement stopping are described in more detail in Section 3.4.5.

### 3.4.2 Expanding Search Terms

*Techniques*
The expert system has several ways of expanding the search terms to reformulate the query. Initially, each user search term is assumed to represent a unique concept which is identified as positive or negative when the query is parsed. To broaden a query, the positive concepts are expanded, one at a time, while the negative concepts are expanded to narrow a query. Search terms are added in order of the closeness of their relationship to the original term; therefore stemwords are added before synonyms, which are added before search terms from outside the thesaurus class. Parent stemgroups are added next, because they represent broader concepts and are therefore most likely to broaden the concept. Sibling stemgroups are added next, since they represent related concepts. Child stemgroups are added last, since they represent more specific, narrower, concepts and are the least likely to broaden the scope of the query.

*Implementation*
The order of firing of the rules for the above techniques is controlled by the goal and reform elements. For each technique, there is a set of four rules which implement it: two to recognize that the technique is the proper one to apply next, a third to broaden the query by applying the technique to positive concepts, and a fourth to narrow the query by applying the same technique to negative concepts.

The first initialization rule fires while the reformulations are continuing in the original, or global, direction. This is the case where control is flowing down one side of Figure 3.2.

The second initialization rule fires when upper and lower bounds on the target number have been established and the system is reformulating in the opposite, local, direction to converge to the target number. Reformulation has progressed too far in the original direction, resulting in either too much broadening or too much narrowing, and now the complementary techniques are being applied to achieve success. This is the case where reformulation has flowed a certain distance down the global side of Figure 3.2 and reformulation is now taking place in the opposite direction. This rule checks that the local reformulations have not used more techniques than were used in the global direction. If this has occurred, context adjustment is the only technique used.

*C Routines*

```
CGetFirstSibling( conceptid, stemid, stemname, distance, frequency ): Boolean
-- This routine gets the first sibling stemgroup for a given concept. It returns true if
-- there is a sibling stemgroup, false otherwise.
    call thesaurus routines to get all the siblings for conceptid;
```

```
    if there was a sibling
            fill global variable with stemids for all siblings;
            fill stemid, stemname, distance, and frequency parameters for first
            sibling;
            return( true );
    else return( false );

CGetNextSibling( conceptid, stemid, stemname, distance, frequency ): Boolean
-- This routine gets the next sibling stemgroup for a given concept. It returns true if
-- there is a sibling stemgroup, false otherwise.
get next sibling from global variable filled by CGetFirstSibling;
if there was another sibling
    fill stemid, stemname, distance, and frequency parameters for the next sibling;
    return( true );
else return( false );
```

*Example Rules to Add Siblings*

```
rule AddSiblingsInit1:
-- This rule fires when  reformulation is continuing in the  global direction, and the
-- last global reformulation technique tried was adding parent stems.
if there is a goal whose type is reformulate, and
    a reform such that the next = global, lastglobal = addparents, lastlocal = original
then
    modify goal.type to be addsiblings;
    modify reform such that lastglobal = addsiblings, laststate = addsiblings, step =
    step + 1, dist = 5;
endrule

rule AddSiblingsInit2:
-- This rule fires when reformulation is continuing in the local direction, and the last
-- local reformulation technique tried was adding parent stems. There is a check
-- that the last global reformulation was not the last local reformulation ( i.e.
-- reformulation got at least this far in the global direction). There is also a check
-- that not only context adjustments are being used to approach the target number.
if there is a goal whose type is reformulate, and
    a reform such that the next = local, lastlocal = addparents, lastglobal <> lastlocal,
    laststate <> adjustcontext
then
    modify goal.type to be addsiblings;
    modify reform such that lastlocal = addsiblings, laststate = addsiblings, step =
    step + 1, dist = 5;
endrule

rule AddPositiveSiblings:
-- This rule adds sibling stems to positive concepts (i.e. it broadens the query).
-- This rule fires once for each positive concept in the query. It expands the
-- concepts in increasing order of frequency. It triggers a rule to remove high
-- frequency stems by setting the goal's subgoal field to freqfilter.
if there is a goal whose type is addsiblings, and whose subgoal is null, and
    a reform such that  next is broaden, and
    a positive concept whose last reformulation was addparents, and
    no positive concept with lower frequency than the above concept group, and
    a current query
```

```
then
    call CGetFirstSibling to get the first sibling stem for the concept group;
    while ( there are siblings to add )
        make a stem element for the new sibling;
        call CGetNextSibling to get the next sibling stem;
    modify goal.subgoal to be freqfilter;
    modify concept.state to be addsiblings;
endrule

rule AddNegativeSiblings:
-- This rule adds sibling stems to negative concepts (i.e. it narrows the query).
-- This rule fires once for each negative concept in the query. It expands the
-- concepts in increasing order of frequency.
if there is a goal whose type is addsiblings, and whose subgoal is null, and
    a reform such that next is narrow, and
    a negative concept whose last reformulation was addparents, and
    no negative concept with lower frequency than the above concept group, and
    a current query
then
    same actions as rule AddPositiveSiblings
endrule
```

*Adding Stemgroups*

The rules which add stemgroups to concepts, such as AddPositiveSiblings, merely identify a set of candidate stemgroups for a given concept. There is a collection of rules which filter these candidates and add the survivors one at a time, evaluating their effect on the query. This process is primarily controlled by the subgoal field of the goal element.

First, a rule fires to remove the high frequency candidates, those occurring more than 250 times in the textbase, from the set. This rule is triggered when the subgoal field of the goal element contains *freqfilter*. Normally, thesauri do not contain high frequency terms, but they appear in this thesaurus as parents of multi-word phrases and need to be filtered out by the expert system. Then, the subgoal field is set to *duplicatefilter* and another rule fires which removes candidates stemgroups that already appear in the query.

Duplicate candidates are kept if their relationship to a user's search term is closer than that of the original instantiation of the stemgroup. Consider the query 'boundary ANDNOT page'. When the stemgroup Page is a candidate for inclusion as a parent stemgroup of Boundary, it is eliminated because Page is already in the query as an original term. This prevents the query from becoming '(Boundary OR Page) ANDNOT Page' which would cancel out the effect of the entire negative part of the query. However, if Segment had been added to Page as a sibling, and later reformulations suggested Segment as a parent of Boundary, this addition would be made since parents have a closer relationship than siblings. The query would become '(Boundary OR Segment) ANDNOT (Page OR Segment)', which would nullify the effect of adding Segment to Page. Since Segment is closer to Boundary than Page, it is not wanted in the negative part of the query.

Finally, rules fire to add the stemgroups that pass through the previous two filters to the concept one at a time. As each is added, the resulting number of passages is determined. (There is also an interactive version of the stem addition rule which asks the user whether or not to include each stem to the given concept.) If the effects of adding the stemgroup are too drastic, the system backtracks by removing the stemgroup. If the result is success, the reformulation process is stopped. Otherwise, the change is kept and the reformulation

continues. When all of the stemgroups for a given concept have been processed, the expert system processes the next concept. When all concepts have been processed, the expert system moves on to the next reformulation technique. This process is diagrammed in Figure 3.3.

### 3.4.3 Adjusting Context

The expert system manipulates four different contexts; it adjusts the distance between words in positive and negative multi-word phrases as well as the distance between positive and negative search concepts. Initially, the positive and negative phrase contexts are set to be one word, with the requirement that the words appear in the same order as in the phrase.

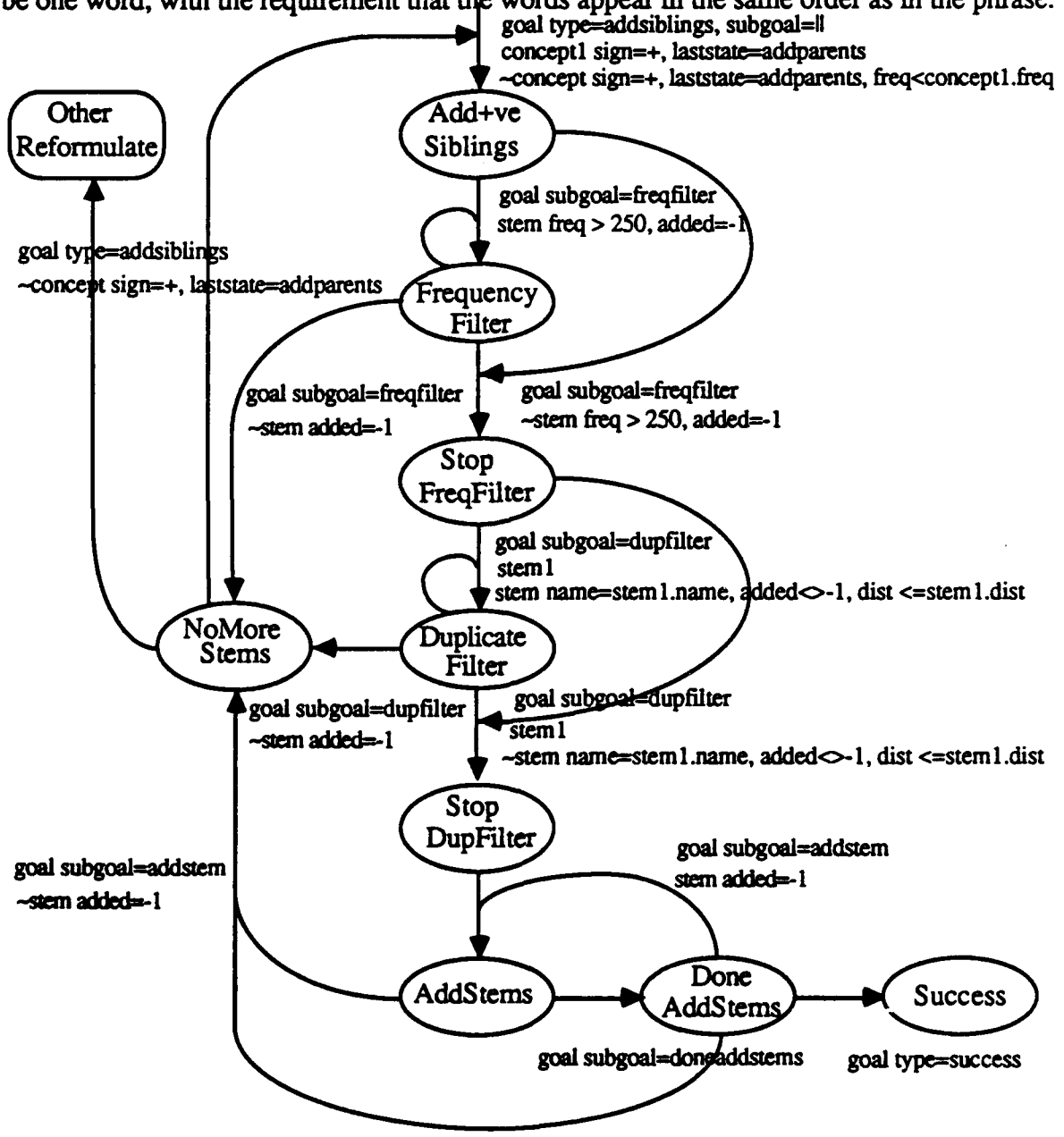



Figure 3.3 Stemgroup Addition Execution Cycle

In other words, the exact phrase is searched for. The default context used for positive and

negative concepts is set to the same sentence. Thus, all positive concepts must appear within the same sentence, and that sentence must not include any occurrences of a negative concept. This context was chosen because it is intermediate between the contextual extremes of adjacency and occurrences within the same paragraph, and it is a natural contextual unit.

Whereas there were several sources of information to draw from to determine the order in which to apply the search term expansion techniques, there was less information available on the use of context to reformulate queries. Contextual searching is not available on standard bibliographic systems, and full-text systems usually supply operators for adjacency, same sentence, same paragraph and same document, only. There is no established practice on when to adjust context rather than expand search terms.

I chose to adjust context in four specific places in the expert system: after adding all stemgroups from the same thesaurus class, but before adding any stemgroups from related thesaurus classes; after adding all related stemgroups from the thesaurus, but before changing the Boolean operators; after changing the Boolean operators but before declaring failure; and after the local reformulations have progressed as far through the search techniques as was used in the original reformulation direction. These places were chosen because it seems desirable to try adjusting context, which is an effective reformulation technique, before moving on to a new group of reformulation techniques which are less promising than the ones already tried.

The first context adjustment occurs after synonyms have been added. If the system is attempting to broaden the query, the context is increased. Conversely, if the system is attempting to narrow the query, the context is decreased. Context is further increased (or decreased) after the children stems have been added. The final increase (or decrease) occurs just before failure is reached (see Figure 3.2). The contexts used, with 1 being the narrrowest and 7 the broadest, are:

1) positive phrases: adjacent words, in correct order
   positive concepts: adjacent words
   negative phrases: within 5 words, any order
   negative concepts: within same paragraph

2) positive phrases: adjacent words, in correct order
   positive concepts: within 3 words
   negative phrases: within 3 words, any order
   negative concepts: within 3 sentences

3) positive phrases: adjacent words, in correct order
   positive concepts: within 5 words
   negative phrases: adjacent words, any order
   negative concepts: within adjacent sentences

4) positive phrases: adjacent words, in correct order
   positive concepts: within same sentence
   negative phrases: adjacent words, in correct order
   negative concepts: within same sentence

5) positive phrases: adjacent words, in any order
positive concepts: within adjacent sentences
negative phrases: adjacent words, in correct order
negative concepts: within 5 words

6) positive phrases: within 3 words, in any order
positive concepts: within 3 sentences
negative phrases: adjacent words, in correct order
negative concepts: within 3 words

7) positive phrases: within 5 words, in any order
positive concepts: within same paragraph
negative phrases: adjacent words, in correct order
negative concepts: adjacent words, any order

Notice that increasing context causes the positive contexts to be broadened while narrowing the negative contexts. The default context appears as number 4.

Besides occurring in the sequence of reformulation techniques, context can be adjusted after any step in the case that local reformulations have progressed as far as the global ones. In this case, the target number has been bracketed, but no query has been close enough to be declared successful. This situation is recognized and context is used to attempt to get closer to the target number, rather than continuing with other, more noisy, techniques. If the context cannot be adjusted to meet the criteria for success, the query which is too broad is chosen over that which is too narrow. Professional searchers tend to present the user with slightly too much information, rather than too little.

### *3.4.4 Changing Boolean Operators*

The final variable the expert system can manipulate to reformulate the user's query is the use of Boolean operators. The query can be broadened in two different ways: first, the AND operators can be switched to OR operators; second, the negative parts of the query can be dropped altogether. Similarly, the query can be narrowed by replacing OR operators with ANDs. The expert system does not have enough information about the user's information needs to decide which positive parts of the query to drop, so this technique is not employed to narrow queries.

Manipulating query structure in this way causes major changes to the user's original query. These techniques are only tried after all close relatives from the thesaurus have been added, and context has been broadened twice. It is not likely that the new query will find passages that the user will find highly relevant, but the goal is to find somewhat relevant passages that users can read in order to reformulate their own queries and try again. It is also possible that there are no passages covering the concepts the user seeks (or far too many), in which case no amount of reformulation could be successful. Showing the final retrieval set, even if it contains no relevant passages at all, indicates to the user this mismatch between query and textbase.

### 3.4.5 Decision Rules

There are a collection of rules to determine the number of passages retrieved by the query sent to MICROARRAS. CheckInitialResults fires only on the user's original query when the system runs without user direction. InteractiveCheckInitialResults fires on the user's original query if the user is to be asked whether or not he wishes to continue the

reformulation. BroadenWhatNext and NarrowWhatNext fire on reformulated queries after a broadening and narrowing step, respectively. These then cause the rules WhatNext or InteractiveWhatNext to fire. I will show the non-interactive versions of CheckInitialResults, BroadenWhatNext, and WhatNext as examples of this family of rules. The rules will be preceded by a description of the C routines they use.

*C Routines*

```
CGetNumPassages( query )
-- Returns the number of passages retrieved by a query.
numpassages = MICROARRAS frequency command( query )
return( numpassages )
CWhatNext( numpassages, toofew, toomany)
-- Returns a symbol representing what the next direction of the reformulation
should be.
if (numpassages < toofew) return( broaden )
    if (numpassages > toomany) return( narrow )
    else return ( success )

CReportDecision( version, numpassages, results)
-- Displays a message to the user informing them of the system's decision on what
to do next.
print( Query version retrieved numpassages passages. Decision: results. )
```

*Sample Decision Rules*

```
rule CheckInitialResults:
--This rule fires to determine the number of passages retrieved by the user's original
-- query. InteractiveCheckInitialResults is identical except that it calls
-- CContinueOrNo to ask the user whether or not to continue reformulating the
-- original query or start over with a new one.
if there is a goal whose type is checkresults and
    a query which has been sent to MICROARRAS and
    no interactive element (indicating that the expert system is running alone) and
    no reform element (indicating that the query is the user's original one)
then
    call CGetNumpassages to determine the number of passages retrieved (store in
        query.numpassages)
    call CWhatNext to determine what the next step should be (store in results)
    call CReportDecision to report back to the user
    if the results were success
        modify goal.type to be success, query.status to be final
    else
        modify goal.type to be reformulate
        modify query.status to be evaluated
        make a reform element with step = 1, global = results, next = results, dist = 0,
            lastglobal = original, lastlocal =original, laststate = original
endrule

rule BroadenWhatNext:
-- This rule fires to determine the number of passages retrieved after a broaden
-- step. It checks that all positive concepts have been reformulated by the previous
-- reformulation technique, and that any candidate stems have been added or
-- discarded. NarrowWhatNext differs from BroadenWhat Next in that it checks
-- for unprocessed negative concepts rather than positive concepts.
```

```
if the current goal is not to reformulate, and its subgoal is null and
    the last reformulation was broadening and
    all positive concepts have been processed for the previous reformulation and
    all candidate stems have been processed for the previous reformulation
then
    modify goal.subgoal to be whatnext
endrule

rule WhatNext:
--This rule fires to decide what to do next based on the number of passages
-- retrieved by a reformulated query. The query's numpassages has already been
-- determined by the reformulation step (it is needed then to decide whether or not
-- to backtrack). InteractiveWhatNext is identical except that it calls
-- CContinueOrNo to ask the user whether or not to continue reformulating the
-- original query or start over with a new one.
if there is a goal whose subgoal is whatnext and
    a query whose number of passages has been determined and
    a reform element (indicating that the query is not the user's original one) and
    no interactive element (indicating that the expert system is running alone)
then
    call CWhatNext to determine what the next step should be (store in results)
    call CReportDecision to report back to the user
    if the results were success
        modify goal.type to be success, query.status to be final
    else
        modify goal.type to be reformulate, goal.subgoal to be null
        modify reform.step to be step + 1, reform.next to be results
endrule
```

### 3.5 Passage Ranking Rules

The dialogue between the expert system and MICROARRAS normally produces a set of passages to be displayed to the user. The last task performed by the expert system is to rank order those passages in terms of their probable interest to the user. To do this, it performs an elementary content analysis on each passage and computes a weight representing probable interest.

Ranking algorithms for document retrieval systems have been extensively studied. There has been less work done on ranking for passage retrieval systems. The FAIR system [Chang and Chow, 1987] performs a simple ranking based on the distance between word pairs, the number of search terms represented and the number of occurrences of the terms. The ranking algorithm used by the expert system considers the following factors: the number of different concepts represented in the passage; the number of different word types for each concept; the relationship of the concept's word types to the user's original search terms; the number of occurrences for each word type from the search expression appearing in the passage; and the contextual distance between search terms. The passages are then ranked according to their respective index values and presented to the user in order of decreasing rank.

#### 3.5.1 Calculating Passage Weights

The weight $W_{pq}$ of passage p for query q, $0 <= W_{pq} <= 1$, is a function of the weight $C_{ip}$ of each query concept i in p, the relationship between the concepts (determined by the

parse tree), and the contextual closeness between the concepts. The concept weights are combined by applying the rules for fuzzy logic [Zadeh, 1965] to the Boolean structure of the query. Additionally, a closeness factor is associated with each of the AND and ANDNOT operators. The closeness factor for the AND operator is set to one of three values (1.0 for same sentence, 0.9 for adjacent sentences, 0.8 for same paragraph). The closer two positive concepts appear in the passage, the higher weight that passage receives. Complementary closeness values are used for the ANDNOT operator (0.8 for same sentence, 0.9 for adjacent sentences, 1.0 for same paragraph).

$$W_{p(Ci\ AND\ Cj)} = \min(Cip, Cjp) * PositiveCloseness \quad (1)$$

$$W_{p(Ci\ OR\ Cj)} = \max(Cip, Cjp) \quad (2)$$

$$W_{p(NOT\ Cj)} = (1 - Cjp) \quad (3)$$

From (1) and (3)

$$W_{p(Ci\ ANDNOT\ Cj)} = \min(Cip, 1 - Cjp) * NegativeCloseness \quad (4)$$

The concept weights and closeness factors fall in the range [0,1], therefore the passage weights also fall in the range [0,1].

### 3.5.2 Calculating Concept Weights

The weight of concept i in passage p, Cip, is a function of the weight of each concept term T in query q, denoted $T_{jq}$ for search term j, and the weight of each concept term in the passage, denoted $T_{jp}$ for search term j, and the number of search terms for the concept. The weight of a search term in the passage is multiplied by the weight of that search term in the query. Thus, the highest weight search terms are those which are important in the query as well as the passage. The weights for all the concept's search terms are summed together and normalized by the number of search terms for the concept, N.

$$C_{ip} = 1/N \sum_{j=1}^{N} T_{jq} * T_{jp} \quad \text{where term j is in concept i} \quad (5)$$

The term weights fall in the range [0,1], therefore, the concept weights also fall in the range [0,1].

### 3.5.3 Calculating Term Weights

Two different term weights, T, are calculated: the weight of the search term i in query q, $T_{iq}$, and the weight of the search term i in passage p, $T_{ip}$.

*Query Term Weights*

The weight of the search term i in query q, $T_{iq}$, reflects the relationship of the search term to the user's original term. The relationships, from closest to most remote, are: same word, stemgroup, synonym, parent, sibling, child. These distances reflect the order in which search terms are added to the concepts, which in turn reflects confidence in the closeness of the relation of the search term to the original term.

$T_{iq}$ = 1.0 (word), 0.9 (stemgroup), 0.8 (synonym), 0.6 (parent), 0.5 (sibling), 0.4 (child) (6)

The query term weights fall in the range [0,1] as required, with the original word receiving a weight of 1.0. Terms added by the expert system receive weights which decrease by 0.1 for every step away from the original term, except for the step from synonym to parent terms. This step decreases the term weight by 0.2, reflecting the large decrease in confidence which occurs when terms are added from outside the thesaurus class.

*Passage Term Weights*
The weight of the search term i in passage p, $T_{ip}$, reflects the frequency of the search term in the passage, $f_{ip}$, and the frequency of the search term in the textbase, $f_{it}$. Jung Soon Ro [Ro, 1988] evaluated several full-text ranking algorithms and concluded that those based on relative document frequency provided the best performance. Thus, I chose relative frequency for the term passage weights.

$$T_{ip} = f_{ip} / f_{it} \quad (7)$$

The term passage weights fall in the range [0,1], as required.

### 3.5.4 Example

As an example of the passage ranking algorithm, the calculation of the weight of a passage retrieved by the sample query of Section 2.6.3 will be presented here. The final version of the query was '(Boundary OR Limit) AND Word ANDNOT page' with the positive context relaxed to adjacent sentences and the negative context restricted to within 5 words. One of the twelve passages retrieved by this query follows. The query term occurrences within the passage are underlined.

> **Need for alignment.** Memory word size affects the efficiency of fetches and stores of information units. If the lengths and locations of such aggregates can be made commensurate with the physical memory word boundaries, such fields are said to be *aligned*, and they can be handled with far greater efficiency than if unaligned. Figure 2-29 illustrates that aligned data require fewer memory accesses than unaligned data.

*Search Terms*
The search terms for the query are:

| | |
|---|---|
| Concept 1: | boundary, boundaries, limit, limits |
| Concept 2: | word, words |
| Concept 3: | page |

*Passage TermWeights*
From Section 3.5.3, the formula for passage term weights is:

$T_{i,p}$ = frequency of term i in passage p/frequency of term i in textbase

Therefore,

$T_{boundaries,p} = 1/9 = 0.1111$

$T_{word,p} = 2/272 = 0.0074$

Since the other terms do not appear in the passage, they have passage term weights of 0.

*Query Term Weights*
As explained in Section 3.5.3, the query term weights are assigned according to the relationship of the term to the user's original search term. Therefore,

| | |
|---|---|
| $T_{boundary,q} = T_{word,q} = T_{page,q} = 1.0$ | (original search term) |
| $T_{boundaries,q} = T_{words,q} = 0.9$ | (same stemgroup) |
| $T_{limit,q} = T_{limits,q} = 0.8$ | (synonym) |

*Concept Weights*
The formula for calculating concept weights, from Section 3.5.2, is:

$$C_{i,p} = 1/N \sum_{j=1}^{N} T_{jq} * T_{jp} \text{ where term j is in concept i}$$

Therefore,

$$C_{1,p} = 1/4 ( (T_{boundary,q} * T_{boundary,p}) + (T_{boundaries,q} * T_{boundaries,p}) + (T_{limit,q} * T_{limit,p}) + (T_{limits,q} * T_{limits,p}) )$$
$$= 1/4 ( (1.0 * 0) + (0.9 * 0.1111) + (0.8 * 0) + (0.8 * 0) )$$
$$= 0.025$$
$$C_{2,p} = 1/2 ( (T_{word,q} * T_{word,p}) + (T_{words,q} * T_{words,p}) )$$
$$= 1/2 ( (1.0 * 0.0074) + (0.9 * 0) )$$
$$= 0.0037$$
$$C_{3,p} = 1/1 ( (T_{page,q} * T_{page,p}) )$$
$$= 1.0 * 0$$
$$= 0$$

*Passage Weight*
The passage weight is calculated from the concept weights as described in Section 3.5.1. The weight for the passage is a factor of the closeness of the concepts. In this case, there are two possible values for closeness; *word* appears in the same sentence as *boundary* as well as in an adjacent sentence. Where there is more than one possible closeness factor, the maximum valid closeness value is used. The parse tree structure is used to combine the concept weights. Since ANDNOT has the highest precedence, the weight for the passage is:

$$W_p(C_1 \text{ AND } C_2 \text{ ANDNOT } C_3)$$
$$= W_p(C_1 \text{ AND } (C_2 \text{ ANDNOT } C_3))$$
$$= \min (C_{1,p}, W_p(C_2 \text{ ANDNOT } C_3)) * \text{PositiveCloseness}$$
$$= \min (C_{1,p}, (\min (C_{2,p}, 1 - C_{3,p}) * \text{NegativeCloseness })) * \text{PositiveCloseness}$$
$$= \min ( 0.025, (\min (0.0037, 1 - 0 ) * 1.0 )) * 1.0$$
$$= \min ( 0.025, 0.0037 ) * 1.0$$
$$= 0.0037$$

The weight of this passage for the reformulated query is 0.0037. This is the weight of the lowest weighted positive concept. If the positive concepts had not occurred in the same sentence, the passage weight would have been reduced by a closeness factor. The weight of the other 11 retrieved passages would be similarly calculated, and the passages would be presented to the user in decreasing weight order. This passage would have one of the highest weights since its weight was not reduced by a closeness factor. Additionally, the terms appearing in the passage had high query term weights as they were closely related to the user's original search terms.

# *Chapter 4*
# *Evaluation*

## 4.1 Introduction

Evaluating an interactive system is difficult. Jean Tague [Tague and Schultz, 1988] has defined a framework for evaluating information retrieval systems. She identified three ways to measure the information retrieval system: informativeness, time, and user friendliness. Informativeness is measured by retrieval output (search effectiveness) and retrieval order (ranking). The search efficiency of the system is related to Tague's time factor. Finally, the user friendliness of the system can be evaluated by a post-search questionnaire.

My primary goal is to demonstrate that using an expert system to reformulate queries can improve search performance for novice searchers. Ideally, both their effectiveness and efficiency would be improved. The second, less important, goal is to show that the expert system can rank the retrieved passages in decreasing order of relevance.

To evaluate the expert system, subjects attempted to find relevant passages in response to high-level questions. They queried MICROARRAS with three interfaces with different capabilities: an interface whose only function was to accept contextual Boolean queries and display search results; a similar interface which also allowed the user to explore the online thesaurus; and a third which incorporated the searching expert system. Each subject's search performance with the three interfaces was monitored and compared.

### 4.1.1 Hypotheses

Hypothesis 1: The expert system improves the search effectiveness for a novice searcher.
Hypothesis 2: The expert system improves the search efficiency for a novice searcher.
Hypothesis 3: The expert system can rank the passages retrieved by the search in decreasing order of relevance.

The effectiveness of the retrieval output is evaluated by looking at recall (the number of relevant items found / the total number of relevant items in the database) and precision (the number of relevant items retrieved / the number of items retrieved). Two estimates of the number of relevant items retrieved are examined: the number of passages the users mark as relevant and the number of passages retrieved from the set of passages deemed relevant by the experimenter.

The efficiency of the systems is measured by the number of Boolean queries the subjects entered for each of several high-level questions, and by the amount of time they spent searching for relevant passages for each question.

The ranking algorithm was evaluated by comparing the order of appearance of relevant passages after they have been ranked with a random order of appearance. Although there is a natural ordering of the passages in the test textbase, i.e. the order of appearance in the book, this is not the case for textbases containing many different, independent documents. Thus, random appearance order was chosen as the control to simulate the expected results when retrieving from a less structured textbase.

These measures, reported separately but considered together, give an overview of the system's performance.

## 4.2 Method

### 4.2.1 Subjects

Twelve computer science graduate students participated as subjects in the study. All subjects were knowledgeable in the use of computers, but unfamiliar with doing online searching. Thus, they were representative of the anticipated users of future information retrieval systems for which this system is a prototype. They might be expected to be better than average searchers since programming ability is highly correlated with searching ability [Borgman,1987]. If the expert system can improve the searching for this subject pool, it might improve performance even more for a user pool with less searching aptitude.

### 4.2.2 Apparatus

*Information Retrieval Systems*
The subjects queried the information retrieval system using three different interfaces. The interfaces were similar in their look and feel, but differed in the functions they provided. The subject sessions were: the user querying MICROARRAS on their own; the user querying MICROARRAS while having access to the online thesaurus; and the user querying MICROARRAS with the help of the expert system.

The *user-alone* configuration consisted of a Sun 3 running MICROARRAS and a rudimentary expert system. This expert system performed only the system control function, and did no query reformulation or ranking of retrieved passages. The user was prompted for a contextual Boolean query, this query was sent to MICROARRAS, and the number of passages retrieved was reported back to the user. The user could display the passages retrieved, if there were fewer than 25, or try another query.

The syntax of the query language, in Backus-Naur form, follows. Non-terminals appear inside angle brackets, and terminals appear in bold.

<query> ::= <term> [<operator> [optional-context] <term>]*
<term> ::= <word> | <stemgroup>
<word> :: = any lower case string
<stemgroup> ::= *<word>
<operator> ::= **and** | **or** | **andnot**
<optional-context> ::= [<context>]
<context> ::= number **to** number <unit> | **nextword** | **sentence** | **paragraph**
<unit> ::= **words** | **sentences** | **paragraphs**

The *user-thesaurus* version consisted of a Sun 3 with one window running MICROARRAS, as in the user-alone system, and a second window running a thesaurus access function. In the thesaurus window, the user had access to all the thesaurus information available to the expert system. He could find out the stemname for a specific word's stemgroup. For any stemname, he could ask for the stemnames of the corresponding synonym, parent, sibling, or child stemgroups. These stemnames could be used in the user's query to MICROARRAS.

The following changes to the query syntax were made in order to allow the user to enter stemnames.

<stemgroup> ::= *<word> | <stemname>
<stemname> ::= any upper case string

In the *user-expert system* version, the user did not have access to the online thesaurus. Context and the addition of stemgroups were controlled by the expert system. Thus, the user entered a Boolean query and a target number of passages, and the expert system reformulated the user's query to attempt to get close to the target number. The user was prompted to filter search terms found in the thesaurus and to continue or abandon the current reformulation. For a fuller description of the user interface, see Section 2.5.

To keep the response time approximately the same as for the other two configurations, it was necessary to run MICROARRAS remotely on the Sun 4 file server containing the textbase. The user worked with one window on a Sun 3 which ran the full version of the query reformulation expert system. The expert system communicated with MICROARRAS over the network. This setup was approximately twice as fast as when MICROARRAS was run on the user's Sun 3. This speed up was necessary, not because the expert system code itself was slow, but rather because the expert system tended to form very long queries involving many MICROARRAS categories and MICROARRAS slows down linearly with the number of search terms in a query.

The changes to the syntax of the query language follows. Note that the user is able to enter a multi-word phrase as a search term.

<query> ::= <term> [<operator> <term>]*
<term> ::= <word>[_<word>]*

*Questions*
Three sets of five questions were devised. Each set contained one training question and four questions on which subjects were monitored. The questions covered material ranging over the whole textbase. They were designed to be specific enough to have on the order of ten relevant passages yet not so specific as to be answerable by a single passage. An estimate of the number of relevant passages in the textbase appears in parentheses after each question. Section 4.2.4 describes how these estimates were calculated.

Query Set A
Practice:
What are some sources of error in floating point arithmetic?
Monitored:
1) How is computer architecture distinguished from the other computer design domains? (16)
2) What are some upward pressures on the level of a machine language? (16)
3) Fixed length multiplication produces a double length result. How have different machines handled this? (14)
4) How are interrupts handled? Do not consider techniques to disable them. (23)

Query Set B
Practice:
I/O devices have moving parts. What is the effect of this motion on the architecture of computers?
Monitored:
1) What are some design principles that lead to clean architectures? Do not consider the economic advantages of a quality design. (14)

2) What techniques have been used to reduce bit traffic? (10)
3) How are control structures implemented? (13)
4) What role does buffering play in I/O transfers? (22)

Query Set C
Practice:
Fragmentation of memory is one problem of using a segmentation scheme. How is paging used to fix this?
Monitored:
1) Discuss the two fundamentally different ways to formally specify an architecture. (19)
2) What are the effects of having two zeros, as in the sign magnitude representation of fixed point numbers? (7)
3) What is done to save state upon a procedure call? (15)
4) Besides I/O, where is concurrency practiced in the implementation? (16)

*Handouts*
The subjects were given three types of handouts. For each version of the retrieval system, they received a description of the appropriate query language including its syntax and some examples. They also received one of the above question sets to answered using the system. When they finished the experiment, they were asked to fill out a questionnaire about the different versions they worked with. A copy of all the handouts appears in Appendix III.

### 4.2.3 Procedure

Subjects received a briefing on the purpose of the experiment and t e type of tasks they would be asked to perform. They were told that they would be querying three versions of the same system. They were asked to try to find on the order of ten relevant passages from the textbase in response to the questions they would be given. They were informed that they might not always be able to find that many, and they were allowed to stop working on a query whenever they were satisfied that they had found as much as they could.

The target number of ten was chosen because it was large enough to require a high recall search, yet small enough that the users would not become tired reading passages. For similar reasons, Carlo Vernimb [Vernimb, 1977] also used a target number of ten when developing an automatic query reformulation system for document retrieval.

Each subject worked with each of the three systems, in turn. This was done to compensate for the large individual differences found in searching ability [Borgman, 1987]. The order of presentation of the three systems was counterbalanced among subjects.

The subjects were coached on query formulation during the training session for the first system. The tools available for query reformulation (i.e. context, search terms, operators) were explained before the subjects used either the user-alone or user-thesaurus version. The query language syntax and system capabilities were explained. The subjects practiced query formulation and reformulation using the training question from set A with their first system. They then queried on the four remaining questions from set A while data describing their session was collected. The subjects then trained and searched with the remaining two systems, in turn, using question set B with their second system and question set C with their third. Between each session they took a short break. When they had completed all three sessions, they were asked to fill out the questionnaire stating their preferences and opinions.

### 4.2.4 Data Collection

*Raw Data*
Data was collected in a trace file while the subjects worked with the system. Each communication from the subject to the retrieval system, and vice versa, was stored with a time stamp. Thus, timing information was collected along with the history of queries entered by the subject and the search results. When the subject chose to display the retrieved passages, those passages and the subject's relevance judgement of them were also stored.

*Data Calculations*
Several parameters were chosen from the trace file to represent each subject's sessions. Measurements were taken on time, number of queries, and number of relevant passages. Before the variables to be compared are described, I will provide a few definitions.

Definitions
A *unique query* was any error-free query entered by a subject. If a subject entered a query which contained a typographic or logical error, and he indicated that he noticed the error by aborting the search and re-entering a corrected version, then the erroneous query was not considered a unique query. However, if the subject gave no indication that he was aware of the error, but instead moved on to a different query altogether, then the erroneous query was considered unique.

The *relevance weight* of a passage is the relevance number assigned to the passage by the subject. A *very relevant (user)* passage is one assigned a relevance weight of two. A *somewhat relevant (user)* passage has a relevance weight of one. A *relevant passage (user)* is one that is either very relevant or somewhat relevant, as judged by the user. An *irrelevant passage (user)* is a passage given a relevance number of zero.

It is necessary to have an estimate of the total number of relevant passages available for each question in order to calculate recall. This estimate was calculated by considering, for each question, the set of all passages judged relevant by any subject. This *absolute retrieval set* contained all very relevant (user) passages minus those I judged irrelevant. In some cases the subjects did not truly understand the question, or the retrieved passages, and marked false drops or marginally relevant passages as very relevant. The tendency to mark marginally relevant passages as very relevant may also be because the subjects were not aware of the presence of much better passages and incorrectly thought that they had retrieved the best passages available. Thus, *relevant passages* are those contained in the absolute retrieval set.

A *successful retrieval set* is a retrieval set containing at least five relevant passages. Since the subjects were attempting to find ten relevant passages, a successful retrieval set contains at least half the number they were looking for. Generally, the size of a successful retrieval set would vary from question to question, with the number of relevant passages available and the user's information needs. However, the questions chosen for this experiment all had between seven and twenty-three relevant passages in the textbase, allowing for a constant target number of ten and a successful retrieval set size of five.

The *final retrieval set* was difficult to identify. Initially, I intended to choose the final set of passages displayed as the final retrieval set. However, on several occasions the subject would have a good retrieval set and try "one last query" which was quite different and retrieved little of relevance. In this case, he would stop searching because he had already achieved satisfactory results. Thus, I choose the last successful retrieval set as the final retrieval set. If a subject never retrieved a successful retrieval set for a given question, the

retrieval set with the highest number of relevant passages, as judged by the subject, was chosen.

Variables

*Total time per question* is calculated from the entry of the subject's first query for the question until after the display, or decision not to display, of the final set of retrieved passages.

*Number of queries per question* is determined by counting the number of unique queries the subject entered for a given question.

*Number of relevant passages (user) found per question* is determined by counting the number of user indicated relevant passages in the final retrieval set for the question.

*User precision* is calculated for the final retrieval set using the standard formula of:
number of relevant passages (user) retrieved / number of passages retrieved

*Number of relevant passages found per question* is determined by counting the number of passages in the final retrieval set for the question that are members of the absolute retrieval set.

*Precision* is calculated for the final retrieval set using the standard formula of
number of relevant passages retrieved (absolute) / number of passages retrieved

*Recall* is calculated for the final retrieval set using the standard formula of
number of relevant passages retrieved (absolute) / total number of relevant passages available

The *ranking balance point* (R) for each retrieval set (not just the final one) is calculated by

$$\frac{\sum_{i=1}^{n} i * \text{relevance}_i}{\sum_{i=1}^{n} \text{relevance}_i}$$

where n = number of passages in the retrieval set
i = position of the passage in the retrieval set
$\text{relevance}_i$ = relevance weight of passage i

This calculates where the midpoint of the relevant passages lies, accounting for the relevance weight. The earlier in the retrieval set the relevant passages occur, the smaller their midpoint. For example, consider a retrieval set of five passages of which the first two are very relevant (weight = 2), the next two irrelevant (weight = 0), and the last passage somewhat relevant (weight = 1). The ranking balance point for this set would be:
(1*2) + (2*2) + (3*0) + (4*0) + (5*1) / 5 = 2.2

The *random balance point* for each retrieval set is calculated by (n+1)/2 where n is the number of passages in the retrieval set. A random distribution of relevant passages in the

set would have the midpoint (M) of the retrieval set as the balance point. Therefore, the random balance point for the set of five passages in the previous example would be 3.

The *best case balance point* (BC) for each retrieval set is calculated by applying the ranking balance point formula to the case where all very relevant passages preceded all somewhat relevant passages which in turn preceded all non-relevant passages in the set. The *worst case balance point* (WC) was not calculated, but it would be equal to M + ( M - BC ), as the range from best case to worst case is symmetric about the random balance point. The best case balance point for the example retrieval set would occur if the somewhat relevant passage appeared in the third, rather than fifth, position. In this case, the ranking balance point would be:

(1*2) + (2*2) + (3*1) + (4*0) + (5*0) / 5 = 1.8

Based on a best case balance point of 1.8, the worst case balance point would be 4.2.

The ranking balance points were normalized so that the random balance point fell on 0, the best case balance point at 1, and the worst case balance point at -1. The normalization performed was:

1) Centre range from best case to worst case about 0 by shifting down M units.
   R' = R - M
2) Set range to be 2, from -1 to +1, by multiplying by 2/range.
   R'' = R' * 2/( BC - WC )
   = R' * 2/(( M - BC ) * 2 ) because range is symmetric about M.
   = ( R - M ) / ( M - BC )
3) Flip sign so that BC = 1, WC = -1.
   Normalized ranking balance point (NR) = ( M - R ) / ( M - BC ).

For the example retrieval set, the normalized ranking balance point would be:

(3 - 2.2) / (3 - 1.8) = 0.67.

Summaries Calculated for Each System

The following results were calculated by entering the numbers generated above into a Microsoft Works spreadsheet (see Appendix IV). For each system the means calculated were:

- number of queries per question
- time per question (seconds)
- number of relevant passages (user) per question
- user precision
- number of relevant passages (from absolute retrieval set)
- precision
- recall

For each ranking algorithm (the expert system's, and randomness) the following means were calculated:

- balance point
- normalized balance point

### 4.3 Results

The means were compared to determine if their differences were statistically significant. Pairwise two-tailed t-tests were performed. A difference was considered significant if its probability of occurring due to chance was less than 5% at the 95% confidence level (a 10% chance at the 95% confidence level was considered marginally significant). Pairs of means with statistically significant differences are flagged with asterisks.

### 4.3.1 Search Effectiveness

All three systems retrieved comparable numbers of relevant passages. Whereas there seemed to be higher recall with the thesaurus, shown by a mean of 7.688 compared to a mean of 7.292 with the expert system, this difference was not significant ($p = 0.5333$).

- number of relevant passages (user) per question
  - user alone 7.375
  - user and thesaurus 7.688
  - user and expert system 7.292

All three systems produced comparable precision, based on the subject's relevance judgements.

- user precision
  - user alone 0.763
  - user and thesaurus 0.786
  - user and expert system 0.761

All three systems retrieved approximately the same number of passages from the absolute retrieval set.

- number of passages from absolute retrieval set
  - user alone 5.521
  - user and thesaurus 5.708
  - user and expert system 5.729

Recall was comparable across all three systems. There was a slight improvement in recall for the user and expert system configuration, but the advantage over the user-alone configuration was not significant ($p < 0.6988$).

- recall
  - user alone 0.364
  - user and thesaurus 0.368
  - user and expert system 0.379

The user and expert system configuration produced marginally significant improvements in precision when compared with the user-alone configuration.

- precision
  - user alone 0.530 * ($p < 0.0817$)
  - user and thesaurus 0.576
  - user and expert system 0.604 *

### 4.3.2 Search Efficiency

The expert system improved search efficiency, as measured by number of user queries over both the user alone and user plus thesaurus.

- number of queries per question
  - user alone 4.833 * ($p < 0.0001$)
  - user and thesaurus 5.458 ** ($p < 0.0001$)
  - user and expert system 2.354 *,**

The expert system was not significantly slower than the other two systems. However, MICROARRAS was being executed by a Sun 4 with the user-expert system configuration resulting in approximately a doubling of its speed. The increase in search time observed when the user was using a thesaurus, rather than searching alone, was significant, but only marginally.

- mean time per question (seconds)
  - user alone 474.5 * ($p < 0.101$)
  - user and thesaurus 571.5 *
  - user and expert system 539.8

### 4.3.3 Ranking

The expert system ranked relevant documents more highly than would be predicted by randomness. The expert system's ranking was compared to a random distribution for 74 sets of retrieved passages.

- balance points
  - random 5.00 * ($p < 0.0165$)
  - expert system 4.53 *
- normalized balance points ( on range of -1 to +1 )
  - random 0.000 * ($p < 0.0025$)
  - expert system 0.195 *

## 4.4 Analysis

The first hypothesis, that the expert system can improve the search effectiveness for a novice user, was partially supported by this study. The expert system produced marginally significant improvements in precision and seemed to indicate improvements in recall, but these results were not significant. Providing the online thesaurus produced no improvement in search effectiveness.

The improvements in precision may result from the expert system applying better broadening techniques. The subjects, when searching alone, would often stop with a very broad query and examine a large set of retrieved passages (over fifteen) looking for relevant information. This type of strategy results in the lower precision observed when the subjects search on their own.

However, this browsing strategy also accounts for the ability of the subjects to produce recall comparable to the expert system. For example, in two questions with large absolute retrieval sets (questions 4 and 12) the subjects were able to retrieve, on average, 10 and 10.25 relevant passages on their own compared with the expert system's retrieval of 8 and 7.75 passages, respectively. By using a target number of 10 for these broader questions, the expert system was operating at a disadvantage. More relevant information was easily found, judging by the high recall of the subjects, but the expert system did not even attempt to further broaden the query.

Another factor which may contribute to the marginal performance improvements seen may be the choice of subjects. The novice searchers in this experiment were all expert programmers. Programming ability has been closely linked with searching ability [Borgman, 1987], so it may be that the subjects in this experiment were able to achieve better than average search results on their own and were less in need of the expert system to search effectively than another group of subjects would be. It would be worthwhile to run a different subject pool with an appropriate textbase to see if the expert system is able to improve the effectiveness of users with lower searching aptitudes.

Finally, an analysis of the expert system's performance on a larger range of questions, and with different stopping criteria, may be able to identify a class of questions for which the existing system outperforms searchers on their own.
The second hypothesis, that the expert system can improve the search efficiency of novice searchers, was supported. Using the expert system significantly reduced the number of queries subjects needed to answer a given question. Subjects required fewer than half as many queries per question on average using the expert system versus the other two versions of the system. This is a substantial improvement. The expert system reduced the amount of user effort required by decreasing the number of queries a user needs to design to express their information needs.

If efficiency is measured in terms of total user time, the expert system fares less well. The expert system was not significantly slower than either of the other two systems, but it was necessary to run MICROARRAS on a faster machine to achieve this. Even running MICROARRAS on a SUN4, the average amount of system time increased from 66 seconds (17%) to 235 seconds (52%). Since the SUN4 configuration was approximately twice as fast, this represents a sevenfold increase in system time. Roughly half of this increase resulted from the increased number of queries MICROARRAS had to handle. The users submitted an average of 2.35 queries per question with the expert system, versus 5 when querying on their own. However, due to extensive reformulation, the expert system sent 15 queries to MICROARRAS per question, on average.

The rest of the difference is caused by the addition of thesaural terms. The increased system time required per query is evenly split between processing the addition of stemgroups and increased MICROARRAS time to evaluate the longer queries formed by the expert system. First, the thesaural terms are retrieved and new MICROARRAS categories are created for each candidate stemgroup. Second, the candidate stemgroups are presented to the user for filtering. Next, the selected stemgroups must be added to the appropriate concept category. Finally, the new, longer, query is sent to MICROARRAS to be evaluated.

The number of queries necessary is a measure of user effort, and I was able to show that the expert system could reduce the amount of work a user had to do. The total user time measures the amount of time a user must spend to get the information he is seeking. This was roughly the same for all three system configurations. The system time was much greater using the expert system, but this is to be expected. This version of the expert system was designed with correctness rather than efficiency in mind, and there are several ways that it could be sped up (see Section 5.2.1).

Allowing the subjects to access the online thesaurus actually decreased the subjects' efficiency. They took significantly more time than when they searched on their own, and required no fewer queries. This allows us to conclude that the improvement in efficiency seen above was due to the expert system's knowledge base, not just the provision of an online thesaurus.

The third hypothesis that the expert system could rank passages in decreasing order of relevance was supported. Although the expert system did present relevant passages significantly earlier than would be predicted by randomness, the improvement was not large enough to be considered truly successful. The current algorithm needs to be evaluated with different weights or a somewhat different algorithm needs to be tried in order to further improve the ranking function. Decreasing the query term weights more quickly as the query terms move farther from the original may improve the ranking by placing more emphasis on the user's original search terms. Using a more sophisticated closeness factor, one that took into account to how many words apart the search terms

were in the passage, as well as sentence and paragraph measures considered in this version, could also lead to improved ranking.

## 4.5 Questionnaire

The subjects were asked for their opinions after they had completed the experiment. This section summarizes their responses.

*User-Alone Version*
The number of subjects (of the total twelve) who said they used each of several reformulation techniques follows. The techniques used by the expert system were tried by most of the subjects. The techniques used by the largest number of subjects - adjusting context, adding stemwords, and adding search terms to concepts - are the same techniques that the expert system tries first.

| | |
|---|---|
| 12 | adjusting context |
| 11 | adding OR terms - from knowledge, from reading passages |
| 7 | replacing terms - from knowledge, from reading passages |
| 8 | removing AND terms |
| 9 | adding ANDNOT terms |
| 12 | using stemwords |
| 8 | whole new query |

*User-Thesaurus Version*
When asked about thesaurus usage, three subjects used it a lot, 4 used it some, and 5 used is very little. Those who used it little or none found it easier to come up with related terms on their own and found using the thesaurus tedious. Subjects were similarly split on the usefulness of the thesaurus, with 6 users finding it helpful and 6 disagreeing. This lack of enthusiasm for the thesaurus is reflected in the experiment by the lack of improvement of effectiveness with the thesaurus and the decrease in efficiency.

*User-Expert System Version*
The subjects were asked which features of the expert system they liked best. The automatic addition of terms from the thesaurus was the most frequently mentioned (8 subjects), whereas the automatic context adjustment was the second most popular feature (3 subjects). Many subjects (8) mentioned the decreased amount of work needed to perform a search, with three of them specifically mentioning that they did not have to think as much. Other features mentioned which decreased the user effort were the simplified syntax, decreased typing, and the fewer queries to remember.

System slowness was the feature most disliked (6 subjects). Although the amount of time necessary to answer a question was no greater with the expert system (see Section 4.3.2), there was less work for the user to do so time seemed longer. The other main complaints concerned the user interface. The subjects were fairly evenly split between wanting the system to proceed more automatically, with less prompting from them (4 subjects), whereas others wanted the system to explain what it was doing and/or allow the user to direct it (5 subjects). These comments lead to the conclusion that if a usable system is to be built based on the success of this research prototype, the execution of the system must be sped up and more work on interface design is needed.

*System Comparisons*
Almost all the subjects (10) found the user-expert system version the easiest to use, with the remaining two subjects split between the other two versions. Not surprisingly, given the comparable effectiveness of the three systems, the subjects were split on which system

they felt gave the best results. Three voted for the user-alone version, two for the user-thesaurus, and three for the expert system. Three said it was a tie between the user-thesaurus and the expert system and one abstained.

Finally, the subjects made many comments the bulk of which dealt with user interface issues. These are summarized in Section 5.1 which deals with possible revisions to the system.

# *Chapter 5*
# *Future Work*

## 5.1 Refinements

Running the experiment suggested several possible refinements to the system. The experimental subjects had many useful comments, the bulk of which dealt with the desire for a more sophisticated user interface. These are summarized in Section 5.2.1. Improvements to the searching knowledge base were suggested by observing the expert system in use. Some of the possible modifications are described in Section 5.1.2.

### 5.1.1 User Interface

*Query Language*
A Boolean query language was used in order to make the system similar to existing retrieval systems. However, the expert system parses the queries to identify positive and negative concepts (each search term is assumed to represent one concept). Since many users have difficulty forming proper Boolean queries (including, surprisingly, some subjects with strong computer science backgrounds in this experiment) it would be easier to have the users list the search terms which represent the concepts directly. The expert system could form initial Boolean queries based on these search terms by joining the original terms with ANDs. The query could then be expanded by adding OR terms from the thesaurus.

*Amount of Interaction*
The desired amount of interaction between the system and the user varies greatly from user to user. The more experienced the searcher, the more user control it may be desirable to allow. On the other hand, many subjects wanted the system to run silently, with fewer prompts for feedback. The two conflicting desires suggest that the amount of interaction should be set by the user to meet his individual needs. Even subjects that indicated they wished more control over the search process did better with the expert system than without. So, although users may wish to direct the system, they may achieve better results if they are not allowed to do so.

There are two ways that the user is asked to interact with the system: filtering suggested stemgroups and deciding whether or not to continue the reformulation. These features are discussed separately in the following paragraphs.

*Stem Filtering*
Users were allowed to choose which stemgroups to add from a list of candidates. This feature was provided in an attempt to improve precision and increase system speed by shortening the queries. There were two problems with allowing the users to filter stemgroups. One is that subjects did not always recognize helpful stemgroups and would filter out the very stemgroups that would have led to relevant passages. The second problem is one of implementation. Rather than filtering at the stemgroup level, users should be shown the relevant section of the thesaurus and allowed to filter entire nodes at a time.

*Stopping Reformulation*
The expert system displays the number of passages retrieved after each reformulation, regardless of whether or not the reformulation caused any change. The subjects found it hard to tell if any progress was being made and tended to abort the reformulation process after several reformulations in a row which had no apparent effect. Often, the reformulation process was stopped before it had run to completion on a query, when in fact a few more reformulations would have led to good search results. It would be better for the expert system to report back only when there has been a change in the retrieval set to prevent users from becoming discouraged.

*Speed*
The desire for more control may be caused simply by the desire to search faster. If the expert system could do its reformulations more quickly, users might have less desire to direct the system themselves in order to "get there quicker." The slowness comes not from the expert system code itself but rather the way in which the expert system interacts with MICROARRAS. The expert system builds larger queries than most users would on their own requiring a lot of work from MICROARRAS to evaluate them. This is an essential characteristic of the expert system which cannot be changed; however the expert system could be more efficient in how it sends the queries to MICROARRAS. Currently, no record is kept of the previous query sent to MICROARRAS and each reformulation causes the query to be evaluated from scratch. All of the categories are redefined although most of them are unchanged from one query to the next. Sending only the changes from one reformulation to the next should make the expert system version more competitive in terms of speed.

*Target Number*
Specifying a target number is an unnatural way for users to state their information needs. It would be preferable to have them indicate more generally the type of search they wish: for example, high recall or high precision. The expert system could use this as a guide to the number of reformulations to try before reporting back to the user. A high recall search would run until completion or until it finds the maximum number of passages anyone is likely to want to read from a screen. The high precision search could stop after adding stemwords, synonyms, and broadening the context, unless there are no passages retrieved using just these reformulations.

### 5.1.2 Knowledge Base

Observing the expert system reformulate real queries gave invaluable insight into which types of queries it handled well, and which it did not. The improvements to the knowledge base suggested by my observations fall into three categories: broadening techniques, narrowing techniques, and multi-word phrases.

*Broadening Techniques*
Currently, the Boolean operators are loosened before negative concepts are removed. It is a bigger change to replace ANDs with ORs than to drop the negative concepts from the query so the order of application of these two reformulation techniques should be swapped.

A common type of query that required broadening was one that contained the intersection of three or more concepts. In this case, broadening context and adding search terms to each concept fails to address the fundamental problem of intersecting too many concepts. The next step of replacing the ANDs with ORs is too drastic a change. It invariably leads to too broad a query. Instead, the expert system should take the original query and drop each of the concepts in turn. For example, the query 'A AND B AND C' would have partial

queries 'A AND B', 'B AND C', and 'A AND C'. The number of passages retrieved by each of the new partial queries should be reported back to the user, and he could restart the expert system on whichever partial query best reflects his interests.

*Narrowing Techniques*
The most common type of query requiring narrowing consisted of a single, high-frequency concept. None of the current reformulation techniques were of any use in this case. There were no operators to change, no context to adjust, and adding search terms just makes the query broader. This type of query should be treated as a special case. The concept's child concepts from the thesaurus should be presented as alternative, more specific, queries. The user could also be encouraged to AND this concept with another.

*Multi-word Phrases*
The treatment of multi-word phrases entered by the user which do not appear in the thesaurus should be changed. Currently, the only expansions done are expanding each word to its stemgroup and loosening the context allowed between the words of the phrase. It would be preferable to treat the words of the phrase as separate concepts which are ANDed together with adjacent context. Each phrase word could then be expanded using the full range of thesaural relationships, as is the case with regular search terms.

## 5.2 New Research Directions

Future research building on this work falls into four major categories: knowledge base, query language, thesaurus, and passage ranking. First, whereas I have successfully developed a searching knowledge base, it is not the only one possible. Other knowledge bases, applying the current search techniques in different orders or containing entirely different sets of search techniques, should be built and compared with the results of this expert system. The goal of this research would be to iterate toward continually improving online search assistants.

Second, more innovative types of query languages should be explored. One possibility that seems interesting is to provide a graphical query language. The user would draw an associative net of concepts. The contents of the nodes would be used to represent the initial search terms and the spatial relationships between the nodes could be used to set the context between the search terms. Furthermore, links between nodes could be used to deduce logical relationships between concepts. This type of query language might work best with a retrieval system based on term weights rather than Boolean queries. In this case, the term weights could also be drawn fro· the spatial relationships between the nodes. It might also be possible to do some graph matching between the user's query graph and the expert system's thesaural graph.

Additionally, research is needed into how to produce the online thesaurus automatically. The Blaauw-Brooks manuscript contains the following hierarchy of concepts: chapter titles, section titles, subsection titles, paragraph titles, and italicized words. A quasi-thesaurus could easily be built using the embedded format marks to identify the word of interest in the original text. The back-of-the-book index could be used to assist identifications of the stemgroups and multi-word phrases. The main difficulties would be handling the many multi-word phrases in the thesaurus and producing the stemgroups. In this case, a stemming algorithm may help. The performance of the expert system using this thesaurus could be compared to the performance using the hand-built one.

Finally, more work is needed to improve the ranking of the retrieved passages. The current ranking algorithm should be tried with different weights for the query search terms and the

closeness factor. It may be necessary to try entirely different algorithms, possibly incorporating syntactic or semantic information, to achieve high quality ranking.

In conclusion, I have demonstrated that an expert system can provide online search assistance to improve the performance of novice searchers. Whereas more research is necessary to develop a better search assistant, I have been able prove that a useful search assistant can be developed which is domain independent, and that implementation of such a system is feasible now.

# *Bibliography*


[Bates, 1979]
Bates, Marcia J., "Information Search Tactics," *Journal of the ASIS*, Vol. 30, No. 4, 1979, pp. 205-214.

[Belew, 1989]
Belew, Richard K., "Adaptive Informtaion Retrieval: Using a Connectionist Representation to Retrieve and Learn About Documents," *Proceedings of the Twelfth Annual International ACMSIGIR Conference on Research & Development in Information Retrieval*, N. J. Belkin and C. J. van Rijsbergen (ed.), ACM Press, 1989, pp. 11-20.

[Belkin et al, 1987]
Belkin, Nicholas J. et al, "Distributed Expert-Based Information Systems: An Interdisciplinary Approach," *Information Processing & Management*, Vol. 23, No. 5, 1987, pp. 387-399.

[Blair and Maron, 1985]
Blair, David C., and Maron, M.E., "An Evaluation of Retrieval Effectiveness for a Full-Text Document-Retrieval System," *Communications of the ACM*, Vol. 28, No. 3, March 1985, pp. 289-299.

[Blaauw and Brooks, 1987]
Blaauw G.A. and Brooks, Frederick P. Jr., *Computer Architecture, Volume 1-Design Decisions*, Draft, Spring 1987.

[Bonzi and Liddy, 1988]
Bonzi, Susan and Liddy, Elizabeth, "The Use of Anaphoric Resolution for Document Description in Information Retrieval," *Proceedings of the Eleventh Annual International ACMSIGIR Conference on Research & Development in Information Retrieval*, Y. Chiaramella (ed.), ACM Press, 1988, pp. 53-66.

[Borgman, 1986]
Borgman, Christine L., "Why Are Online Catalogs Hard to Use?," *Journal of the ASIS*, Vol. 37, No. 6, November 1986, pp. 387-400.

[Borgman, 1987]
Borgman, Christine L., "Individual Differences in the Use of Information Retrieval Systems: Some Issues and Some Data," *Proceedings of the Tenth Annual International ACMSIGIR Conference on Research & Development in Information Retrieval*, C.J. von Rijsbergen and C.T. Yu (ed.), ACM Press, 1987, pp. 61-69.

[Brajnik et al, 1988]
Brajnik, G., Guida, G., and Tasso, C., "IR-NLI II: Applying Man-Machine Interaction and Artificial Intelligence Concepts to Information Retrieval," *Proceedings of the Eleventh Annual International ACMSIGIR Conference on Research & Development in Information Retrieval*, Y. Chiaramella (ed.), ACM Press, 1988, pp. 387-399.

[Brooks, 1983]
Brooks, Helen M., *Informatics 7: Intelligent Information Retrieval*, Kevin P. Jones (ed.), 1983, pp. 65-78.

[Chang and Chow, 1987]
Chang, and Chow, "Towards a Friendly Adaptable Information Retrtieval System , *Proceedings of RIAO 88*, M.I.T., March 1987, pp. 172-182.

[Chiaramella and Defude, 1987]
Chiaramella, Y. and Defude, B., "A Prototype of and Intelligent System for Information Retrieval: IOTA," *Information Processing & Management*, Vol. 23, No. 4, 1987, pp. 285-303.

[Croft and Lewis, 1987]
Croft, W. Bruce, and Lewis, David D., "An Approach to Natural Language Processing for Document Retrieval," *Proceedings of the Tenth Annual International ACMSIGIR Conference on Research & Development in Information Retrieval*, C.J. von Rijsbergen and C.T. Yu (ed.), ACM Press, 1987, pp. 26-32.

[Croft et al, 1988]
Croft, W. B., Lucia, T. J., Cohen P. R., "Retrieving Documents by Plausible Inference: A Preliminary Study," *Proceedings of the Eleventh Annual International ACMSIGIR Conference on Research & Development in Information Retrieval*, Y. Chiaramella (ed.), ACM Press, 1988, pp. 481-494.

[Croft and Thompson, 1987]
Croft, W. B. and Thompson, R. H., "I$^3$R: A New Approach to the Design of Document Retrieval Systems,"*Journal of the ASIS*, Vol. 38, No. 6, November 1987, pp. 389-404.

[Crouch, 1988]
Crouch, Carolyn J., "A Cluster-Based Approach to Thesaurus Construction," *Proceedings of the Eleventh Annual International ACMSIGIR Conference on Research & Development in Information Retrieval*, Y. Chiaramella (ed.), ACM Press, 1988, pp. 309-320.

[Driscoll et al, 1988]
Driscoll, James R., Rajala, David A, Shaffer, William H., and Thomas, Donald W., "An Application of Artificial Intelligence Techniques to Automated Keywording," *Proceedings of RIAO 88*, M.I.T., March 1988, pp. 500-511.

[Fenichel, 1981]
Fenichel, C. H., "Online Searching: Measures that Discriminate Among Users with Different Types of Experience," *Journal of the ASIS*, Vol. 32, No. 1, January 1981, pp. 23-32.

[Fox et al, 1988]
Fox, Edward A., Weaver, Marybeth T., Chen, Qi-Fan, and France, Robert K., "Implementing a Distributed Expert-Based Information Retrieval System," *Proceedings of RIAO 88*, M.I.T., March 1988, pp. 708-726.

[Giger, 1988]
Giger, H. P., "Concept Based Retrieval in Classical IR Systems," *Proceedings of the Eleventh Annual International ACMSIGIR Conference on Research & Development in Information Retrieval*, Y. Chiaramella (ed.), ACM Press, 1988, pp. 275-289.

[Green et al, 1969]
Green, B. F., Wolf, A. K., Chomsky, C., and Laughery, K., "BASEBALL: An automatic question answerer," in *Computers and Thought*, McGraw-Hill, New York, 1969, pp. 207-216.

[Harman, 1988]
Harman, Donna, "Towards Interactive Query Expansion," *Proceedings of the Eleventh Annual International ACMSIGIR Conference on Research & Development in Information Retrieval*, Y. Chiaramella (ed.), ACM Press, 1988, pp. 321-331.

[Jacobs and Rau, 1988]
Jacobs, Paul S. and Rau, Lisa F., "Natural Language Techniques for Intelligent Information Retrieval," *Proceedings of the Eleventh Annual International ACMSIGIR Conference on Research & Development in Information Retrieval*, Y. Chiaramella (ed.), ACM Press, 1988, pp. 85-99.

[Katz, 1988]
Katz, Boris, "Using English for Indexing and Retrieving," *Proceedings of RIAO 88*, M.I.T., March 1988, pp. 314-332.

[Krovetz and Croft, 1989]
Krovetz, Robert and Croft, W. Bruce, "Word Sense Disambiguation using Machine-Readable Dictionaries," *Proceedings of the Twelfth Annual International ACMSIGIR Conference on Research & Development in Information Retrieval*, N. J. Belkin and C. J. van Rijsbergen (ed.), ACM Press, 1989. pp. 127-136.

[Lebowitz, 1985]
Lebowitz, Michael, "RESEARCHER: An Experimental Intelligent Information System," *Proceedings of the 9th IJCAI*, Vol. 2., 1985, pp. 858-862.

[Ledwith, 1988]
Ledwith, Robert H., "Development of a Large Concept Oriented Database for Informaiton Retrieval," *Proceedings of the Eleventh Annual International ACMSIGIR Conference on Research & Development in Information Retrieval*, Y. Chiaramella (ed.), ACM Press, 1988, pp. 651-661.

[Marcus, 1981]
Marcus, Richard S., "An Automated Assistant for Information Retrieval," *Proceedings of the 44th ASIS Annual Meeting*, Vol. 18, 1981, pp. 270-273.

[Marcus, 1986]
Marcus, Richard S., "Design Questions in the Development of Expert Systems for Retrieval Assistance," *Proceedings of the 49th ASIS Annual Meeting*, Vol. 23, 1986, pp. 185-189.

[McCune et al, 1985]
McCune, Brian P., Tong, Richard M., Dean, Jeffrey S., Shapiro, Daniel G., "RUBRIC: A System for Rule-Based Information Retrieval," IEEE Transactions on *Software Engineering*, Vol. SE-11, No. 9, September 1985, pp. 939-945.

[Meadow et al, 1982]
Meadow, Charles T., Hewett, Thomas T., and Aversa, Elizabeth S., "A Computer Intermediary for Interactive Database Searching. I. Design," *Journal of the ASIS*, Vol. 33, No. 4, July 1982, pp. 325-332.

[Metzler and Haas, 1989]
Metzler, Douglas P. and Haas, Stephanie W., "The Constituent Object Parser: Syntactic Structure Matching for Information Retrieval," *Proceedings of the Twelfth Annual International ACMSIGIR Conference on Research & Development in Information Retrieval*, N. J. Belkin and C. J. van Rijsbergen (ed.), ACM Press, 1989. pp. 117-126.

[Oddy, 1977]
Oddy, R.N., "Information Retrieval Through Man-Machine Dialogue," *Journal of Documentation*, Vol. 33, No. 1, 1977, pp. 1-14.

[Oldroyd, 1984]
Oldroyd, B. K., "Study of strategies used in online searching 5: differences between the experienced and the inexperienced searcher," *Online Review*, Vol. 8, No. 3, 1984, pp. 233-244.

[Pollitt, 1984]
Pollitt, A.S., "A `Front-End' System: An Expert System as an Online Search Intermediary," *ASLIB Proceedings*, Vol. 36, No. 5, May 1984, pp. 229-234.

[Pollitt, 1987]
Pollitt, A.S., "CANSEARCH: An Expert Systems Approach to Document Retrieval," *Information Processing & Management*, Vol. 23, No. 2, 1987, pp. 119-136.

[Prasse et al, 1988]
Prasse, M.J., Dillon, M., Gordon, M.J., Mortland, B., and Repka, A., "F-TAS: A Full-Text Access System," *Presented at the National Online Meeting*, New York City, May 10-12, 1988.

[Ro, 1988]
Ro, Jung Soon, "An Evaluation of the Applicability of Ranking Algorithms to Improve the Effectiveness of Full-Text Retrieval. II. On the Effectiveness of Ranking Algorithms on Full-Text Retrieval," *Journal of the ASIS*, Vol. 39, No. 3, May 1988, pp. 147-160.

[Salton and Buckley, 1988]
Salton, Gerard, and Buckley, Chris, "Parallel Text Search Methods," *Communications of the ACM*, Vol. 31, No. 2, February 1988, pp. 202-215.

[Salton and Smith, 1989]
Salton, Gerard and Smith, Maria, "On the Application of Syntactic Methodologies in Automatic Text Analysis," *Proceedings of the Twelfth Annual International ACMSIGIR Conference on Research & Development in Information Retrieval*, N. J. Belkin and C. J. van Rijsbergen (ed.), ACM Press, 1989. pp. 137-151.

[Salton and McGill, 1983]
Salton, G. and McGill, M.J., *Introduction to Modern Information Retrieval*, New York: McGraw-Hill, 1983.

[Smeaton and van Rijsbergen, 1983]
Smeaton, A. F., and van Rijsbergen, C. J., "The Retrieval Effects of Query Expansion on a Feedback Document Retrieval System," *The Computer Journal*, Vol. 26, No. 3, 1983, pp. 239-246.

[Smeaton and van Rijsbergen, 1988]
Smeaton, A. F., and van Rijsbergen, C. J., "Experiments on Incorporating Syntactic Processing of User Queries into a Document Retrieval Strategy," *Proceedings of the Eleventh Annual International ACMSIGIR Conference on Research & Development in Information Retrieval*, Y. Chiaramella (ed.), ACM Press, 1988, pp. 31-51.

[Smith et al, 1987]
Smith, John B., Weiss, Stephen F., and Ferguson, Gordon J., "MICROARRAS: An Advanced Full-Text Retrieval and Analysis System," *Proceedings of the Tenth Annual International ACMSIGIR Conference on Research & Development in Information Retrieval*, C.J. von Rijsbergen and C.T. Yu (ed.), ACM Press, 1987, pp. 187-195.

[Smith et al, 1989]
Smith, Phillip J., Shute, Steven J., and Galdes, Deb, *Proceedings of the Twelfth Annual International ACMSIGIR Conference on Research & Development in Information Retrieval*, N. J. Belkin and C. J. van Rijsbergen (ed.), ACM Press, 1989. pp. 3-10.

[Sparck Jones, 1983]
Sparck Jones, Karen, "Intelligent Retrieval," *Informatics 7: Intelligent Information Retrieval*, Kevin P. Jones (ed.), 1983, pp. 136-143.

[Teskey, 1987]
Teskey, Niall, "Extensions to the Advanced Interface Management Project," *OCLC Research Review*, July 1987, pp. 1-3.

[Tong et al, 1987]
Tong, Richard M., Applebaum, Lee A., Askmann, Victor N., and Cunningham, James F., "Conceptual Information Retrieval using RUBRIC," *Proceedings of the Tenth Annual International ACMSIGIR Conference on Research & Development in Information Retrieval*, C.J. von Rijsbergen and C.T. Yu (ed.), ACM Press, 1987, pp. 247-253.

[Vernimb, 1977]
Vernimb, Carlo, "Automatic Query Adjustment in Document Retrieval," *Information Processing & Management*, Vol. 13, No. 6, 1977, pp. 339-353.

[Vickery and Brooks, 1987]
Vickery, A., and Brooks, Helen M., "PLEXUS-The Expert System for Referral," *Information Processing & Management*, Vol. 23, No. 2, 1987, pp. 99-117.

[Williams, 1984]
Williams, P.W., "A Model for an Expert System for Automated Information Retrieval," *Proceedings of the 8th International Online Meeting*, 1984, pp. 139-149.

[Zadeh, 1965]
Zadeh, L. A., "Fuzzy Sets," *Information and Control*, Vol. 8, 1965, pp. 338-353.

[Zarri, 1984a]
Zarri, Gian Piero, "Expert Systems and Information Retrieval: An Experiment in the Domain of Biographical Data Management," *International Journal of Man-Machine Studies*, Vol. 20, 1984, pp. 87-106.

[Zarri, 1984b]
Zarri, Gian Piero, "Some Remarks about the Inference Techniques of RESEDA, an 'Intelligent' Information Retrieval System," *Research and Development in IR: Proceedings of the 3rd Joint BCS and ACM Symposium*, C.J. von Rijsbergen (ed.), Cambridge University Press, 1984, pp. 281-300.

# Appendix I
# Table of Contents for Textbase

# Appendix II
# Rule Base

## II.1 OUtils.ops

This module contains the type declarations, external procedure definitions, working memory element declarations, and the local procedures.

```
module OUtils ()
{

-- Type declarations
type buffer = array( 1025:char );

-- The external C routines used by the Expert System.
external procedure CAddNegativeChildren( &OldQuery: out integer, &NewQuery: out
    integer);
external procedure CAddNegativeParents( &OldQuery: out integer, &NewQuery: out
    integer);
external procedure CAddNegativeSiblings( &OldQuery: out integer, &NewQuery: out
    integer);
external procedure CAddNegativeStemwords( &OldQuery: out integer, &NewQuery: out
    integer);
external procedure CAddNegativeSynonyms( &OldQuery: out integer, &NewQuery: out
    integer );
external procedure CAddPositiveChildren( &OldQuery: out integer, &NewQuery: out
    integer);
external procedure CAddPositiveParents( &OldQuery: out integer, &NewQuery: out
    integer);
external procedure CAddPositiveSiblings( &OldQuery: out integer, &NewQuery: out
    integer);
external procedure CAddPositiveStemwords( &OldQuery: out integer, &NewQuery: out
    integer);
external procedure CAddPositiveSynonyms( &OldQuery: out integer, &NewQuery: out
    integer );
external procedure CChangeContext( &OldContext: out integer, &NewContext: out integer
    );
external procedure CAddStem( &Concept: out integer, &StemId: out integer );
external function CContinueOrNo(): logical;
external procedure CCopyBuffer( &Destination: out buffer, &Source: out buffer );
external function CDisplayOrNo( &Query: out integer,&NumPassages: out integer ):
    symbol;
external procedure CDisplayPassage( &Passage: out integer );
external procedure CDisplayConcordance( &Reply: out symbol );
external procedure CErrorHandle( &Message: out symbol );
external function CFillStemgroup( &Conceptid: out integer, &Stemid: out integer, &Dist:
    out integer ): integer;
external function CGetNewConcept( &Id: out integer, &Sign: out char, &Stem: out
    integer, &Freq: out integer ): logical;
external function CGetFirstPassage( &Reply: out symbol,&Passage: out integer ): logical;
external function CGetNextPassage( &Reply: out symbol, &Passage: out integer ): logical;
external function CGetFirstChild( &Concept: out integer, &Stem: out integer, &Word: out
```

```
    symbol, &Dist: out integer, &Freq: out integer ): logical;
external function CGetNextChild( &Concept: out integer, &Stem: out integer, &Word: out
    symbol, &Dist: out integer, &Freq: out integer ): logical;
external function CGetFirstParent( &Concept: out integer, &Stem: out integer, &Word: out
    symbol, &Dist: out integer, &Freq: out integer ): logical;
external function CGetNextParent( &Concept: out integer, &Stem: out integer, &Word: out
    symbol, &Dist: out integer, &Freq: out integer ): logical;
external function CGetFirstSibling( &Concept: out integer, &Stem: out integer, &Word:
    out symbol, &Dist: out integer, &Freq: out integer ): logical;
external function CGetNextSibling( &Concept: out integer, &Stem: out integer, &Word:
    out symbol, &Dist: out integer, &Freq: out integer ): logical;
external function CGetFirstSynonym( &Concept: out integer, &Stem: out integer, &Word:
    out symbol, &Dist: out integer, &Freq: out integer ): logical;
external function CGetNextSynonym( &Concept: out integer, &Stem: out integer, &Word:
    out symbol, &Dist: out integer, &Freq: out integer ): logical;
external function CGetNumPassages( &Reply: out symbol ): integer;
external procedure CGetRelevance();
external function CGetQuery( &Version: out integer, &Target: out integer ): logical;
external function CGetPassageWt( &Passage: out integer ): real;
external procedure CGetStemName( &Id: out integer, &Name: out symbol );
external procedure CGetUniqueName( &FileName: out symbol );
external function CKeepChange( &OldNum: out integer, &NewNum: out integer, &Min:
    out integer, &Target: out integer, &Max: out integer ): symbol;
external procedure CKillSystem();
external procedure CLoosenOps( &OldQuery: out integer, &NewQuery: out integer );
external procedure CTightenOps( &OldQuery: out integer, &NewQuery: out integer );
external procedure CPassQueryToMA( &Query: out integer, &Reply: out symbol );
external procedure CRemoveAndnots( &OldQuery: out integer, &NewQuery: out integer );
external procedure CRemoveConcept( &Concept: out integer );
external procedure CRemoveQuery( &Query: out integer );
external procedure CRemoveStem( &Concept: out integer, &StemId: out integer);
external procedure CReportDecision(&Query: out integer, &NumPassages: out integer,
    &Decision: out symbol );
external function CStartSystem( &TraceFile: out integer ): logical;
external procedure CUnFillStemgroup( &Conceptid: out integer, &Stemid: out integer );
external function CUserWantStem( &Conceptid: out integer, &Stemname: out symbol):
    logical;
external function CWhatNext( &NumPassages: out integer, &Min: out integer, &Max: out
    integer): symbol;
external procedure CWtPassages( &Query: out integer, &NumPassages: out integer,
    &Reply: out symbol, &DisplayName: out symbol );

-- The working memory element type definitions.
type start = element();
type interactive = element();    -- is user involved in reformulation
```

```
type goal = element
(
  type : symbol;
  subgoal : symbol;
  min : integer;        -- the minimum acceptable number of passages
  toofew : integer;     -- the lower bound on "success"
  justright : integer;  -- the target number
  toomany : integer;    -- the upper bound on "success"
  max : integer;        -- the maximum acceptable number
);

type reform = element
(
  global : symbol;      -- narrow, broaden, or adjustcontext
  next : symbol;        -- narrow, broaden, or ||
  lastglobal : symbol;  -- last global state visited
  lastlocal : symbol;   -- last local state visited
  laststate : symbol;   -- last state visited
  step : integer;       -- counter of steps in reformulation
  dist : integer;       -- distance of new terms from original term
);

type query = element
(
  status : symbol;
  version : integer;
  name : symbol;
  numpassages: integer;
  context : integer;
  hi : integer;
  lo : integer;
);

type concept = element
(
  status : symbol;      -- is concept in current query or not
  id : integer;         -- unique identifier (index to C structure)
  sign : char;          -- + or -
  freq : integer;       -- total frequency of concept group members
  state : symbol;       -- last broaden/narrow operation
);

type stem = element
(
  id : integer;         -- unique identifer (index to C structure)
  concept : integer;    -- concept group that term is a member of
  name : symbol;        -- stemname
  freq : integer;       -- frequency of the stemgroup
  added : integer;      -- which reformulation added the term
  dist : integer;       -- distance of stem from original term
                        -- (0=same; 1=stemgroup; 2=synonym; 3=parent;
                        --  4=sibling; 5=child)
);
```

```
type passage = element
(
  status : symbol;
  id : integer;
  wt : real;
);

-- Constants
-- As OPS does not have named constants, I am using 20 = maxcontext,
-- 23=defaultcontext, 26=maxcontext for now.

-- Global Variables
global
  &TRACE : integer;

-- Procedures
--
-- NameToSymbol() is called by the C routines to convert an array of
-- characters into a symbol. It is necessary because C routines cannot
-- directly call cvsymbol.
--
function NameToSymbol(&N : name, &S : out symbol) : logical
{
  return( cvsymbol(&S, &N) );
};

--
-- SymbolToName() is called by the C routines to convert a symbol into
-- an array of characters. It is necessary because C routines cannot
-- directly call name.
--
function SymbolToName(&S: symbol, &N : out name) : integer
{
  return( name(&N, &S) );
};

--
-- Quit returns OTRUE if the command is Quit, OFALSE otherwise.
--
simple function Quit(&Q : buffer) : logical
{
  if ( &Q[1] = 'A' ∧ &Q[2] = 'O' ) return( 1B )
  else return( 0B );
};

};
```

## II.2 ORules.ops

This module contains the production rules which form the heart of the knowledge base.

```
module ORules ()
{

use OUtils;

-- Goal: start ----------------------------------------------------

rule Init
{
  &I (start);
-->
local &tracefile:integer;

  &tracefile = &TRACE;
  if ( CStartSystem( &tracefile ) = 0B )
    call CErrorHandle( |Cannot start system.| )
  else
    make ( goal type=getquery; subgoal=||;
             min=-1; toofew=-1; justright=-1; toomany=-1; max=-1 );
  remove &I;
};

-- Goal: getquery -------------------------------------------------
--
-- Get a query from the user.
--
rule GetQuery
{
  &G ( goal type=getquery );
-->
local &querynum : integer,
    &target : integer,
    &realtarget : real,
    &id : integer,
    &sign : char,
    &freq : integer,
    &stemid : integer,
    &name : symbol,
    &min : integer,
    &toofew : integer,
    &toomany : integer,
    &max : integer;

  &querynum = 0;
  if ( CGetQuery( &querynum, &target ) = 0B )
    modify &G ( type=quit )
  else
  {
    make ( query status=new; version=&querynum; context=23; hi=23; lo=23;
             numpassages=-1; name=|| );
```

```
    &id = 0;
    while ( CGetNewConcept( &id, &sign, &stemid, &freq ) = 1B )
    {
      call CGetStemName( &stemid, &name );
      make ( concept id=&id; sign=&sign; freq=&freq; status=active;
                state=original );
      make ( stem id=&stemid; concept=&id; name=&name; freq=&freq;
               added=0; dist=0 );
      &id = &id+1;
    };

    cast &realtarget = &target;
    cast &min = &realtarget*0.5;
    if ( &min < 1 ) &min=1;

    cast &toofew = &realtarget*0.8;
    if ( &toofew < 1 ) &toomany=1;

    cast &toomany = &realtarget*1.2;
    if ( &toomany > 25 ) &toomany=25;

    cast &max = &realtarget*1.5;
    if ( &max > 25 ) &max=25;

    modify &G ( type=sendquerytoMA; min=&min; toofew=&toofew;
           justright=&target; toomany=&toomany; max=&max );
  };
};

-- Goal: sendquery ------------------------------------------------
--
-- Send a query to MICROARRAS.
--
rule SendQueryToMA
{
  &G ( goal type=sendquerytoMA );
  &Q ( query status=new );
-->
local &version:integer;
local &name:symbol;

  &version = &Q.version;
  call CPassQueryToMA( &version, &name );

  modify &G ( type=checkresults );
  modify &Q ( status=sent; name=&name );
};
```

```
-- Goal: checkresults ----------------------------------------------------
--
-- Find out how many passages were retrieved by the user's intiial query.
-- If reformulation is necessary, this rule creates a reform wme, and sets
-- the global and last fields.
--
rule CheckInitialResults
{
  &G ( goal type=checkresults );
  ~ ( interactive );
  &Q ( query status=sent );
  ~ ( reform )
-->
local &name : symbol;
local &numpassages : integer;
local &version : integer;
local &results : symbol;
local &toofew : integer;
local &target : integer;
local &toomany : integer;

  &name = &Q.name;
  &version = &Q.version;
  &numpassages = CGetNumPassages( &name );
  &toofew = &G.toofew;
  &target = &G.justright;
  &toomany = &G.toomany;
  &results = CWhatNext( &numpassages, &toofew, &toomany );
  call CReportDecision( &version, &numpassages, &results );

  if ( &results=success )
  {
    modify &G ( type=success );
    modify &Q ( status=final; numpassages=&numpassages );
  }
  else
  {
    modify &G ( type=reformulate );
    modify &Q ( status=evaluated; numpassages=&numpassages );
    make ( reform step=1; global=&results; next=&results; dist=0;
           lastglobal=original; lastlocal=original; laststate=original )
  };
};

rule InteractiveCheckInitialResults
{
  &G ( goal type=checkresults );
  &I ( interactive );
  &Q ( query status=sent );
  ~ ( reform )
-->
local &name : symbol;
local &numpassages : integer;
local &version : integer;
```

```
local &results : symbol;
local &toofew : integer;
local &target : integer;
local &toomany : integer;
local &continue : logical;

  &name = &Q.name;
  &version = &Q.version;
  &numpassages = CGetNumPassages( &name );
  &toofew = &G.toofew;
  &target = &G.justright;
  &toomany = &G.toomany;
  &results = CWhatNext( &numpassages, &toofew, &toomany );
  call CReportDecision( &version, &numpassages, &results );
  &continue = CContinueOrNo();

  if (( &results=success ) V ( &continue=0B ))
  {
    modify &G ( type=success );
    modify &Q ( status=final; numpassages=&numpassages );
  }
  else
  {
    modify &G ( type=reformulate );
    modify &Q ( status=evaluated; numpassages=&numpassages );
    make ( reform step=1; global=&results; next=&results; dist=0;
             lastglobal=original; lastlocal=original; laststate=original )
  };
};

--
-- After all concepts have been processed, ask the user for direction.
--
rule BroadenWhatNext
{
  &G ( goal type<>reformulate; subgoal=ll );
  &R ( reform next=broaden );
  ~ ( concept sign='+'; state<>&R.laststate );
  ~ ( stem added=-1 );
-->
  modify &G ( subgoal=whatnext );
};

--
-- After all concepts have been processed, ask the user for direction.
--
rule NarrowWhatNext
{
  &G ( goal type<>reformulate; subgoal=ll );
  &R ( reform next=narrow );
  ~ ( concept sign='-'; state<>&R.laststate );
  ~ ( stem added=-1 );
-->
```

```
  modify &G ( subgoal=whatnext );
};


-- SubGoal: whatnext ----------------------------------------
--
-- Decide what to do next based on the target number.
--
rule WhatNext
{
  &G ( goal subgoal=whatnext );
  ~ ( interactive );
  &Q ( query status=evaluated );
  &R ( reform );
-->
local &results : symbol;
local &version : integer;
local &numpassages : integer;
local &toofew : integer;
local &target : integer;
local &toomany : integer;

  &numpassages = &Q.numpassages;
  &version = &Q.version;
  &toofew = &G.toofew;
  &target = &G.justright;
  &toomany = &G.toomany;
  &results = CWhatNext( &numpassages, &toofew, &toomany );
  call CReportDecision( &version, &numpassages, &results );

  if ( &results=success )
  {
    modify &G ( type=success; subgoal=ll );
    modify &Q ( status=final );
    remove &R;
  }
  else
  {
    modify &G ( type=reformulate; subgoal=ll );
    modify &R ( step=@.step+1; next=&results );
  };
};

--
-- Decide what to do next based on the target number.
--
rule InteractiveWhatNext
{
  &G ( goal subgoal=whatnext );
  &I ( interactive );
  &Q ( query status=evaluated );
  &R ( reform );
-->
```

```
local &results : symbol;
local &version : integer;
local &numpassages : integer;
local &toofew : integer;
local &target : integer;
local &toomany : integer;
local &continue : logical;

  &numpassages = &Q.numpassages;
  &version = &Q.version;
  &toofew = &G.toofew;
  &target = &G.justright;
  &toomany = &G.toomany;
  &results = CWhatNext( &numpassages, &toofew, &toomany );
  call CReportDecision( &version, &numpassages, &results );
  &continue = CContinueOrNo();

  if (( &results=success ) V ( &continue=0B ))
  {
    modify &G ( type=success; subgoal=|| );
    modify &Q ( status=final );
    remove &R;
  }
  else
  {
    modify &G ( type=reformulate; subgoal=|| );
    modify &R ( step=@.step+1; next=&results );
  };
};



-- SubGoal: aredonefillstems ----------------------------------------
--
-- Decide whether or not we have success, based on the target number.
-- Used only in the middle of the fillstems loop.
--
rule AreDoneFillStems
{
  &G ( goal subgoal=aredonefillstems );
  &Q ( query status=evaluated );
  &R ( reform );
-->
local &results : symbol;
local &version : integer;
local &numpassages : integer;
local &toofew : integer;
local &target : integer;
local &toomany : integer;

  &numpassages = &Q.numpassages;
  &version = &Q.version;
  &toofew = &G.toofew;
  &target = &G.justright;
```

```
  &toomany = &G.toomany;
  &results = CWhatNext( &numpassages, &toofew, &toomany );

  if ( &results=success )
  {
    call CReportDecision( &version, &numpassages, &results );
    modify &G ( type=success );
    modify &Q ( status=final );
    remove &R;
  };
  modify &G ( subgoal=ll );
};

-- SubGoal: aredoneaddstems ----------------------------------------
--
-- Decide whether or not we have success, based on the target number.
-- Used only in the middle of the addstems loop.
--
rule AreDoneAddStems
{
  &G ( goal subgoal=aredoneaddstems );
  &Q ( query status=evaluated );
  &R ( reform );
-->
local &results : symbol;
local &version : integer;
local &numpassages : integer;
local &toofew : integer;
local &target : integer;
local &toomany : integer;

  &numpassages = &Q.numpassages;
  &version = &Q.version;
  &toofew = &G.toofew;
  &target = &G.justright;
  &toomany = &G.toomany;
  &results = CWhatNext( &numpassages, &toofew, &toomany );

  if ( &results=success )
  {
    call CReportDecision( &version, &numpassages, &results );
    modify &G ( type=success );
    modify &Q ( status=final );
    remove &R;
  };
  modify &G ( subgoal=addstems );
};
```

```
-- SubGoal: filterstems ----------------------------------------------
--
-- Remove any stems with frequency > maxfrequency.
--
rule FrequencyFilter
{
  &G ( goal subgoal=freqfilter );
  &S ( stem freq > 250; added=-1 );
-->
  remove &S ;
};

--
-- Go on to duplicate filtering.
--
rule FrequencyFilterStop
{
  &G ( goal subgoal=freqfilter );
  ~ ( stem freq > 250; added=-1 );
-->
  modify &G ( subgoal=dupfilter );
};

--
-- If there are no stems left, bypass dupfilter
--
rule NoStemsToAdd1
{
  &G ( goal subgoal=freqfilter );
  ~ ( stem added=-1 );
-->
  modify &G ( subgoal=ll );
};

--
-- Remove any duplicate stems
--
rule DuplicateFilter
{
  &G ( goal subgoal=dupfilter );
  &S1 ( stem added=-1 );
  &S2 ( stem (@.name=&S1.name); added<>-1; (@.dist <= &S1.dist) );
-->
  remove &S1;
};

--
-- No more filtering to be done.
--
rule DuplicateFilterStop
{
  &G ( goal subgoal=dupfilter );
  &S ( stem added=-1 );
  ~ ( stem (@.name=&S.name); added<>-1; (@.dist <= &S.dist) );
```

```
-->
  modify &G ( subgoal=addstems );
};

--
-- If there are no stems left, bypass addstems
--
rule NoStemsToAdd2
{
  &G ( goal subgoal=dupfilter );
  ~ ( stem added=-1 );
-->
  modify &G ( subgoal=ll );
};

-- SubGoal: fillstem ----------------------------------------------------
--
-- Fill stems one at a time, checking for effect on query response.
--
rule FillStem
{
  &G ( goal subgoal=fillstem );
  &Q ( query status=evaluated );
  &S ( stem added=-1 );
  &C ( concept id=&S.concept );
  &R ( reform )
-->
local &freq : integer;
local &stemid : integer;
local &conceptid : integer;
local &name : symbol;
local &numpassages : integer;
local &oldnum : integer;
local &dist : integer;
local &min, &target, &max : integer;
local &decision : symbol;

  &stemid = &S.id;
  &conceptid = &C.id;
  &dist = &R.dist;

  &oldnum = &Q.numpassages;
  &min = &G.min;
  &target = &G.justright;
  &max = &G.max;

  -- Fill in the stemgroup
  &freq = CFillStemgroup( &conceptid, &stemid, &dist );

  -- See how many passages there are now.
  &name = &Q.name;
  &numpassages = CGetNumPassages( &name );
```

```
  -- Check that the new stem doesn't put us outside the acceptable range.
  &decision = CKeepChange( &oldnum, &numpassages, &min, &target, &max );
  if ( (&decision = |keep|) )
  {
    modify &S ( freq=(@.freq+&freq); dist=1 );
    modify &C ( freq=(&S.freq) );
    modify &Q ( numpassages=&numpassages );
  };

  if ( &decision = |discard| )
    call CUnFillStemgroup( &conceptid, &stemid );

  modify &S ( added=&R.step );
  modify &G ( subgoal=aredonefillstems );
};

-- SubGoal: addstems ----------------------------------------------------
--
-- Check with user before adding the stem.
--
rule InteractiveAddStem
{
  &G ( goal subgoal=addstems );
  &I ( interactive );
  &Q ( query status=evaluated );
  &S ( stem added=-1 );
  ~ ( stem (@.freq < &S.freq); added=-1 );
  &C ( concept id=&S.concept );
  &R ( reform )
-->
local &concept : integer;
local &freq : integer;
local &stemid : integer;
local &stemname : symbol;
local &oldnum : integer;
local &numpassages : integer;
local &name : symbol;
local &min, &target, &max : integer;
local &decision : symbol;

  &concept = &S.concept;
  &stemname = &S.name;
  if ( CUserWantStem( &concept, &stemname ) = 1B )
  {
    &stemid = &S.id;

    &oldnum = &Q.numpassages;
    &min = &G.min;
    &target = &G.justright;
    &max = &G.max;

    call CAddStem( &concept, &stemid );
```

```
    &name = &Q.name;
    &numpassages = CGetNumPassages( &name );

    -- Check that the new stem doesn't put us outside the acceptable range.
    &decision = CKeepChange( &oldnum, &numpassages, &min, &target, &max );
    if ( &decision = |keep| )
    {
      modify &S ( added=&R.step );
      modify &C ( freq=(@.freq+&S.freq) );
      modify &Q ( numpassages=&numpassages );
    };
    if ( &decision = |discard| )
    {
      call CRemoveStem( &concept, &stemid );
      remove &S;
    };
  }
  else
    remove &S;

  modify &G ( subgoal=aredoneaddstems );
};

--
-- Add new stems one at a time, in increasing order of frequency,
-- checking for effect on query response.
--
rule AddStem
{
  &G ( goal subgoal=addstems );
  ~ ( interactive );
  &Q ( query status=evaluated );
  &S ( stem added=-1 );
  ~ ( stem (@.freq < &S.freq); added=-1 );
  &C ( concept id=&S.concept );
  &R ( reform )
-->
local &concept : integer;
local &freq : integer;
local &stemid : integer;
local &oldnum : integer;
local &numpassages : integer;
local &name : symbol;
local &min, &target, &max : integer;
local &decision : symbol;

  &concept = &S.concept;
  &stemid = &S.id;

  &oldnum = &Q.numpassages;
  &min = &G.min;
  &target = &G.justright;
  &max = &G.max;
```

```
  call CAddStem( &concept, &stemid );

  &name = &Q.name;
  &numpassages = CGetNumPassages( &name );

  -- Check that the new stem doesn't put us outside the acceptable range.
  &decision = CKeepChange( &oldnum, &numpassages, &min, &target, &max );
  if ( &decision = |keep| )
  {
    modify &S ( added=&R.step );
    modify &C ( freq=(@.freq+&S.freq) );
    modify &Q ( numpassages=&numpassages );
  };

  if ( &decision = |discard| )
  {
    call CRemoveStem( &concept, &stemid );
    remove &S;
  };
  modify &G ( subgoal=aredoneaddstems );
};

--
-- After all stems for this concept have been added, process the next concept.
--
rule NoMoreStemsToAdd
{
  &G ( goal subgoal=addstems );
  &Q ( query status=evaluated );
  ~ ( stem added=-1 );
  &R ( reform );
-->
  modify &G ( subgoal=|| );
};


-- SubGoal: changecontext ----------------------------------------------
--
-- Increase the context.
--
rule IncreaseContext
{
  &G ( goal subgoal=changecontext );
  &R ( reform next=broaden );
  &Q ( query status=evaluated );
-->
local &oldcontext : integer;
local &newcontext : integer;
local &name : symbol;
local &oldnum : integer;
local &numpassages : integer;
local &min, &target, &max : integer;
local &decision : symbol;
```

```
  &oldnum = &Q.numpassages;
  &min = &G.min;
  &target = &G.justright;
  &max = &G.max;

  &oldcontext = &Q.context;
  &newcontext = &oldcontext+1;
  call CChangeContext( &oldcontext, &newcontext );

  &name = &Q.name;
  &numpassages = CGetNumPassages( &name );

  -- Check that the new stem doesn't put us outside the acceptable range.
  &decision = CKeepChange( &oldnum, &numpassages, &min, &target, &max );
  if ( &decision = |keep| )
  {
    modify &Q ( numpassages=&numpassages; context=&newcontext );
    if ( &newcontext > &Q.hi )
      modify &Q ( hi=&newcontext );
  };

  if ( &decision = |discard| )
    call CChangeContext( &newcontext, &oldcontext );

  modify &G ( subgoal=whatnext );
};

--
-- Decrease the context.
--
rule DecreaseContext
{
  &G ( goal subgoal=changecontext );
  &R ( reform next=narrow );
  &Q ( query status=evaluated );
-->
local &oldcontext : integer;
local &newcontext : integer;
local &name : symbol;
local &oldnum : integer;
local &numpassages : integer;
local &min, &target, &max : integer;
local &decision : symbol;

  &oldcontext = &Q.context;
  &newcontext = &oldcontext-1;
  call CChangeContext( &oldcontext, &newcontext );

  &oldnum = &Q.numpassages;
  &min = &G.min;
  &target = &G.justright;
  &max = &G.max;
```

```
  &name = &Q.name;
  &numpassages = CGetNumPassages( &name );

  -- Check that the new stem doesn't put us outside the acceptable range.
  &decision = CKeepChange( &oldnum, &numpassages, &min, &target, &max );
  if ( &decision = |keep| )
  {
    modify &Q ( numpassages=&numpassages; context=&newcontext );
    if ( &newcontext < &Q.lo )
      modify &Q ( lo=&newcontext );
  };

  if ( &decision = |discard| )
    call CChangeContext( &newcontext, &oldcontext );

  modify &G ( subgoal=whatnext );
};

-- Goal: reformulate ----------------------------------------------------

-- addstemwords ---------------------------------------------------------
--
-- Add stemwords to the query if we are continuing in the global direction
-- and the last global state is original.
--
rule AddStemwordsInit1
{
  &G ( goal type=reformulate );
  &R ( reform (@.global=@.next); lastglobal=original; lastlocal=original );
-->
  modify &G ( type=addstemwords );
  modify &R ( lastglobal=addstemwords; laststate=addstemwords; step=@.step+1;
          dist=1 );
};

--
-- Add stemwords to the query if we are continuing in the local direction
-- and the last local state is original, and the last global state is not original
--
rule AddStemwordsInit2
{
  &G ( goal type=reformulate );
  &R ( reform (@.global<>@.next); lastlocal=original; (@.lastglobal<>@.lastlocal);
laststate<>adjustcontext );
-->
  modify &G ( type=addstemwords );
  modify &R ( lastlocal=addstemwords; laststate=addstemwords; step=@.step+1;
          dist=1 );
};
```

```
--
-- Add stemwords to the positive stemgroups, in decreasing order of
-- frequency of concept group, if we are broadening.
--
rule AddPositiveStemwords
{
  &G ( goal type=addstemwords; subgoal=ll );
  &R ( reform next=broaden );
  &C ( concept sign='+'; state=original );
  ~ ( concept sign='+'; state=original; (@.freq < &C.freq) );
  &S ( stem concept=&C.id );
-->
  modify &S ( added=-1 );
  modify &C ( state=addstemwords );
  modify &G ( subgoal=fillstem );
};

--
-- Add stemwords to the negative stemgroups, in decreasing order of
-- frequency of concept group, if we are narrowing.
--
rule AddNegativeStemwords
{
  &G ( goal type=addstemwords; subgoal=ll );
  &R ( reform next=narrow );
  &C ( concept sign='-'; state=original );
  ~ ( concept sign='-'; state=original; (@.freq < &C.freq) );
  &S ( stem concept=&C.id );
-->
  modify &S ( added=-1 );
  modify &G ( subgoal=fillstem );
  modify &C ( state=addstemwords );
};

-- addsynonyms ----------------------------------------------------------
--
-- Add synonyms to the query if we are continuing in the global direction
-- and the last global state is original.
--
rule AddSynonymsInit1
{
  &G ( goal type=reformulate );
  &R ( reform (@.global=@.next); lastglobal=addstemwords; lastlocal=original );
-->
  modify &G ( type=addsynonyms );
  modify &R ( lastglobal=addsynonyms; laststate=addsynonyms; step=@.step+1;
        dist=2 );
};
```

```
--
-- Add synonyms to the query if we are continuing in the local direction
-- and the last local state is original, and the last global state is not original
--
rule AddSynonymsInit2
{
  &G ( goal type=reformulate );
  &R ( reform (@.global<>@.next); lastlocal=addstemwords;
(@.lastglobal<>@.lastlocal); laststate<>adjustcontext );
-->
  modify &G ( type=addsynonyms );
  modify &R ( lastlocal=addsynonyms; laststate=addsynonyms; step=@.step+1;
        dist=2 );
};

--
-- Add synonyms to the positive concept groups, if broadening.
--
rule AddPositiveSynonyms
{
  &G ( goal type=addsynonyms; subgoal=ll );
  &R ( reform next=broaden );
  &C ( concept sign='+'; state=addstemwords );
  ~ ( concept sign='+'; state=addstemwords; (@.freq < &C.freq) );
  &Q ( query status=evaluated );
-->
local &conceptid : integer;
local &stemid : integer;
local &freq : integer;
local &name : symbol;
local &more : logical;
local &dist : integer;

  &dist = &R.dist;
  &conceptid = &C.id;
  &more = CGetFirstSynonym( &conceptid, &stemid, &name, &dist, &freq );
  while ( &more )
  {
    make ( stem id=&stemid; concept=&conceptid; name=&name;
        freq=&freq; added=-1; dist=2 );
    &more = CGetNextSynonym( &conceptid, &stemid, &name, &dist, &freq );
  };

  modify &G ( subgoal=freqfilter );
  modify &C ( state=addsynonyms );
};

--
-- Add synonyms to the negative concept groups, if narrowing.
--
rule AddNegativeSynonyms
{
  &G ( goal type=addsynonyms; subgoal=ll );
  &R ( reform next=narrow );
```

```
  &C ( concept sign='-'; state=addstemwords );
  ~ ( concept sign='-'; state=addstemwords; (@.freq < &C.freq) );
  &Q ( query status=evaluated );
-->
local &conceptid : integer;
local &stemid : integer;
local &freq : integer;
local &name : symbol;
local &more : logical;
local &dist : integer;

  &dist = &R.dist;
  &conceptid = &C.id;
  &more = CGetFirstSynonym( &conceptid, &stemid, &name, &dist, &freq );
  while ( &more )
  {
    make ( stem id=&stemid; concept=&conceptid; name=&name;
         freq=&freq; added=-1; dist=2 );
    &more = CGetNextSynonym( &conceptid, &stemid, &name, &dist, &freq );
  };

  modify &G ( subgoal=freqfilter );
  modify &C ( state=addsynonyms );
};

-- change context1 ----------------------------------------------------------
--
-- Change the context if we are continuing in the global direction,
-- and the last global state was addsynonyms.
--
rule ChangeContextInit1
{
  &G ( goal type=reformulate );
  &R ( reform (@.global=@.next); lastglobal=addsynonyms; lastlocal=original );
-->
  modify &G ( type=changecontext1; subgoal=changecontext );
  modify &R ( lastglobal=changecontext1; laststate=changecontext1; step=@.step+1 );
};

--
-- Change Context if we are continuing in the local direction and the
-- last local state was addsynonyms, and the last global step was
-- not adding synonyms.
--
rule ChangeContextInit2
{
  &G ( goal type=reformulate );
  &R ( reform (@.global<>@.next); lastlocal=addsynonyms;
(@.lastglobal<>@.lastlocal); laststate<>adjustcontext );
-->
  modify &G ( type=changecontext1; subgoal=changecontext );
  modify &R ( lastlocal=changecontext1; laststate=changecontext1; step=@.step+1 );
};
```

```
-- addparents ----------------------------------------------------------
--
-- Add parents to the query if we are continuing in the global direction
-- and the last global state is original.
-- As we are changing the query contents, reset lo and hi.
--
rule AddParentsInit1
{
  &G ( goal type=reformulate );
  &R ( reform (@.global=@.next); lastglobal=changecontext1; lastlocal=original );
  &Q ( query status=evaluated );
-->
  modify &G ( type=addparents );
  modify &R ( lastglobal=addparents; laststate=addparents; step=@.step+1;
          dist=4 );
  modify &Q ( lo=@.context; hi=@.context );
};

--
-- Add parents to the query if we are continuing in the local direction
-- and the last local state is original, and the last global state is not original
--
rule AddParentsInit2
{
  &G ( goal type=reformulate );
  &R ( reform (@.global<>@.next); lastlocal=changecontext1;
(@.lastglobal<>@.lastlocal); laststate<>adjustcontext );
-->
  modify &G ( type=addparents );
  modify &R ( lastlocal=addparents; laststate=addparents; step=@.step+1;
          dist=4 );
};

--
-- Add parents to the positive concept groups, if broadening.
--
rule AddPositiveParents
{
  &G ( goal type=addparents; subgoal=ll );
  &R ( reform next=broaden );
  &C ( concept sign='+'; state=addsynonyms );
  ~ ( concept sign='+'; state=addsynonyms; (@.freq < &C.freq) );
  &Q ( query status=evaluated );
-->
local &conceptid : integer;
local &stemid : integer;
local &freq : integer;
local &name : symbol;
local &more : logical;
local &dist : integer;

  &dist = &R.dist;
  &conceptid = &C.id;
  &more = CGetFirstParent( &conceptid, &stemid, &name, &dist, &freq );
```

```
  while ( &more )
  {
    make ( stem id=&stemid; concept=&conceptid; name=&name;
         freq=&freq; added=-1; dist=3 );
    &more = CGetNextParent( &conceptid, &stemid, &name, &dist, &freq );
  };

  modify &G ( subgoal=freqfilter );
  modify &C ( state=addparents );
};

--
-- Add parents to the negative concept groups, if narrowing.
--
rule AddNegativeParents
{
  &G ( goal type=addparents; subgoal=ll );
  &R ( reform next=narrow );
  &C ( concept sign='-'; state=addsynonyms );
  ~  ( concept sign='-'; state=addsynonyms; (@.freq < &C.freq) );
  &Q ( query status=evaluated );
-->
local &conceptid : integer;
local &stemid : integer;
local &freq : integer;
local &name : symbol;
local &more : logical;
local &dist : integer;

  &dist = &R.dist;
  &conceptid = &C.id;
  &more = CGetFirstParent( &conceptid, &stemid, &name, &dist, &freq );
  while ( &more )
  {
    make ( stem id=&stemid; concept=&conceptid; name=&name;
         freq=&freq; added=-1; dist=3 );
    &more = CGetNextParent( &conceptid, &stemid, &name, &dist, &freq );
  };

  modify &G ( subgoal=freqfilter );
  modify &C ( state=addparents );
};

-- addsiblings ----------------------------------------------------------
--
-- Add siblings to the query if we are continuing in the global direction
-- and the last global state is original.
--
rule AddSiblingsInit1
{
  &G ( goal type=reformulate );
  &R ( reform (@.global=@.next); lastglobal=addparents; lastlocal=original );
-->
```

```
  modify &G ( type=addsiblings );
  modify &R ( lastglobal=addsiblings; laststate=addsiblings; step=@.step+1;
          dist=5 );
};

--
-- Add siblings to the query if we are continuing in the local direction
-- and the last local state is original, and the last global state is not original
--
rule AddSiblingsInit2
{
  &G ( goal type=reformulate );
  &R ( reform (@.global<>@.next); lastlocal=addparents; (@.lastglobal<>@.lastlocal);
laststate<>adjustcontext );
-->
  modify &G ( type=addsiblings );
  modify &R ( lastlocal=addsiblings; laststate=addsiblings; step=@.step+1;
          dist=5 );
};

--
-- Add siblings to the positive concept groups, if broadening.
--
rule AddPositiveSiblings
{
  &G ( goal type=addsiblings; subgoal=ll );
  &R ( reform next=broaden );
  &C ( concept sign='+'; state=addparents );
  ~ ( concept sign='+'; state=addparents; (@.freq < &C.freq) );
  &Q ( query status=evaluated );
-->
local &conceptid : integer;
local &stemid : integer;
local &freq : integer;
local &name : symbol;
local &more : logical;
local &dist : integer;

  &dist = &R.dist;
  &conceptid = &C.id;
  &more = CGetFirstSibling( &conceptid, &stemid, &name, &dist, &freq );
  while ( &more )
  {
    make ( stem id=&stemid; concept=&conceptid; name=&name;
         freq=&freq; added=-1; dist=4 );
    &more = CGetNextSibling( &conceptid, &stemid, &name, &dist, &freq );
  };

  modify &G ( subgoal=freqfilter );
  modify &C ( state=addsiblings );
};
```

```
--
-- Add siblings to the negative concept groups, if narrowing.
--
rule AddNegativeSiblings
{
  &G ( goal type=addsiblings; subgoal=ll );
  &R ( reform next=narrow );
  &C ( concept sign='-'; state=addparents );
  ~ ( concept sign='-'; state=addparents; (@.freq < &C.freq) );
  &Q ( query status=evaluated );
-->
local &conceptid : integer;
local &stemid : integer;
local &freq : integer;
local &name : symbol;
local &more : logical;
local &dist : integer;

  &dist = &R.dist;
  &conceptid = &C.id;
  &more = CGetFirstSibling( &conceptid, &stemid, &name, &dist, &freq );
  while ( &more )
  {
    make ( stem id=&stemid; concept=&conceptid; name=&name;
         freq=&freq; added=-1; dist=4 );
    &more = CGetNextSibling( &conceptid, &stemid, &name, &dist, &freq );
  };

  modify &G ( subgoal=freqfilter );
  modify &C ( state=addsiblings );
};

-- addchildren ----------------------------------------------------------
--
-- Add children to the query if we are continuing in the global direction
-- and the last global state is original.
--
rule AddChildrenInit1
{
  &G ( goal type=reformulate );
  &R ( reform (@.global=@.next); lastglobal=addsiblings; lastlocal=original );
-->
  modify &G ( type=addchildren );
  modify &R ( lastglobal=addchildren; laststate=addchildren; step=@.step+1;
          dist=6 );
};
```

```
--
-- Add children to the query if we are continuing in the local direction
-- and the last local state is original, and the last global state is not original
--
rule AddChildrenInit2
{
  &G ( goal type=reformulate );
  &R ( reform (@.global<>@.next); lastlocal=addsiblings; (@.lastglobal<>@.lastlocal);
laststate<>adjustcontext );
-->
  modify &G ( type=addchildren );
  modify &R ( lastlocal=addchildren; laststate=addchildren; step=@.step+1;
          dist=6 );
};

--
-- Add children to the positive concept groups, if broadening.
--
rule AddPositiveChildren
{
  &G ( goal type=addchildren; subgoal=ll );
  &R ( reform next=broaden );
  &C ( concept sign='+'; state=addsiblings );
  ~ ( concept sign='+'; state=addsiblings; (@.freq < &C.freq) );
  &Q ( query status=evaluated );
-->
local &conceptid : integer;
local &stemid : integer;
local &freq : integer;
local &name : symbol;
local &more : logical;
local &dist : integer;

  &dist = &R.dist;
  &conceptid = &C.id;
  &more = CGetFirstChild( &conceptid, &stemid, &name, &dist, &freq );
  while ( &more )
  {
    make ( stem id=&stemid; concept=&conceptid; name=&name;
         freq=&freq; added=-1; dist=5 );
    &more = CGetNextChild( &conceptid, &stemid, &name, &dist, &freq );
  };

  modify &G ( subgoal=freqfilter );
  modify &C ( state=addchildren );
};
```

```
--
-- Add children to the negative concept groups, if narrowing.
--
rule AddNegativeChildren
{
  &G ( goal type=addchildren; subgoal=ll );
  &R ( reform next=narrow );
  &C ( concept sign='-'; state=addsiblings );
  ~ ( concept sign='-'; state=addsiblings; (@.freq < &C.freq) );
  &Q ( query status=evaluated );
-->
local &conceptid : integer;
local &stemid : integer;
local &freq : integer;
local &name : symbol;
local &more : logical;
local &dist : integer;

  &dist = &R.dist;
  &conceptid = &C.id;
  &more = CGetFirstChild( &conceptid, &stemid, &name, &dist, &freq );
  while ( &more )
  {
    make ( stem id=&stemid; concept=&conceptid; name=&name;
         freq=&freq; added=-1; dist=5 );
    &more = CGetNextChild( &conceptid, &stemid, &name, &dist, &freq );
  };

  modify &G ( subgoal=freqfilter );
  modify &C ( state=addchildren );
};

-- changecontext2 ----------------------------------------------------------
--
-- Change the context if we are continuing in the global direction,
-- and the last global state was addchildren.
--
rule ChangeContextInit3
{
  &G ( goal type=reformulate );
  &R ( reform (@.global=@.next); lastglobal=addchildren );
-->
  modify &G ( type=changecontext2; subgoal=changecontext );
  modify &R ( lastglobal=changecontext2; laststate=changecontext2; step=@.step+1 );
};
```

```
--
-- Change Context if we are continuing in the local direction and the
-- last local state was addchildren, and the last global step was
-- not adding synonyms.
--
rule ChangeContextInit4
{
  &G ( goal type=reformulate );
  &R ( reform (@.global<>@.next); lastlocal=addchildren; (@.lastglobal<>@.lastlocal) );
-->
  modify &G ( type=changecontext2; subgoal=changecontext );
  modify &R ( lastlocal=changecontext2; laststate=changecontext2; step=@.step+1 );
};

-- replaceops ------------------------------------------------------
--
-- Replace the boolean operators if we are continuing in the global direction,
-- and the last global state was changing context.
-- As we are changing the query contents, reset lo and hi.
--
rule ReplaceOpsInit1
{
  &G ( goal type=reformulate );
  &R ( reform (@.global=@.next); lastglobal=changecontext2; lastlocal=original );
  &Q ( query status=evaluated );
-->
  modify &G ( type=replaceops );
  modify &R ( lastglobal=replaceops; laststate=replaceops; step=@.step+1 );
  modify &Q ( lo=@.context; hi=@.context );
};

--
-- Replace the boolean operators if we are continuing in the local direction
-- and the last local state was changecontext2, and the last global step was
-- not adding synonyms.
--
rule ReplaceOpsInit2
{
  &G ( goal type=reformulate );
  &R ( reform (@.global<>@.next); lastlocal=changecontext2;
(@.lastglobal<>@.lastlocal); laststate<>adjustcontext );
-->
  modify &G ( type=replaceops );
  modify &R ( lastlocal=replaceops; laststate=replaceops; step=@.step+1 );
};

--
-- Loosen the boolean operators.
--
rule LoosenOps
{
  &G ( goal type=replaceops; subgoal=ll );
  &R ( reform next=broaden );
  &Q ( query status=evaluated );
```

```
-->
local &oldquerynum : integer;
local &newquerynum : integer;
local &oldcontext : integer;
local &name : symbol;
local &oldnum : integer;
local &numpassages : integer;
local &min, &target, &max : integer;
local &decision : symbol;

  &oldnum = &Q.numpassages;
  &min = &G.min;
  &target = &G.justright;
  &max = &G.max;

  &oldquerynum = &Q.version;
  &newquerynum = &oldquerynum+1;
  call CLoosenOps( &oldquerynum, &newquerynum );

  &oldcontext = &Q.context;
  call CChangeContext( &oldcontext, 23 );

  call CPassQueryToMA( &newquerynum, &name );
  &numpassages = CGetNumPassages( &name );

  -- Check that the new stem doesn't put us outside the acceptable range.
  &decision = CKeepChange( &oldnum, &numpassages, &min, &target, &max );
  if ( &decision = |keep| )
    modify &Q ( status=evaluated; version=&newquerynum;
        context=23; hi=23; lo=23; numpassages=&numpassages; name=&name  );

  if ( &decision = |discard| )
  {
    call CRemoveQuery( &newquerynum );
    call CChangeContext( 23, &oldcontext );
  };

  modify &G ( subgoal=whatnext );
};

--
-- Tighten the boolean operators.
--
rule TightenOps
{
  &G ( goal type=replaceops; subgoal=|| );
  &R ( reform next=narrow );
  &Q ( query status=evaluated );
-->
local &oldquerynum : integer;
local &newquerynum : integer;
local &oldcontext : integer;
local &name : symbol;
local &oldnum : integer;
```

```
local &numpassages : integer;
local &min, &target, &max : integer;
local &decision : symbol;

  &oldnum = &Q.numpassages;
  &min = &G.min;
  &target = &G.justright;
  &max = &G.max;

  &oldquerynum = &Q.version;
  &newquerynum = &oldquerynum+1;
  call CTightenOps( &oldquerynum, &newquerynum );

  &oldcontext = &Q.context;
  call CChangeContext( &oldcontext, 23 );

  call CPassQueryToMA( &newquerynum, &name );
  &numpassages = CGetNumPassages( &name );

  -- Check that the new stem doesn't put us outside the acceptable range.
  &decision = CKeepChange( &oldnum, &numpassages, &min, &target, &max );
  if ( &decision = |keep| )
    modify &Q ( status=evaluated; version=&newquerynum;
         context=23; hi=23; lo=23; numpassages=&numpassages; name=&name );

  if ( &decision = |discard| )
  {
    call CRemoveQuery( &newquerynum );
    call CChangeContext( 23, &oldcontext );
  };

  modify &G ( subgoal=whatnext );
};

-- removenots ------------------------------------------------------------
--
-- Remove andnots if we are continuing to broaden globally,
-- and the last broaden step was replacing the boolean operators.
--
rule RemoveNotsInit1
{
  &G ( goal type=reformulate );
  &R ( reform global=broaden; next=broaden; lastglobal=replaceops; lastlocal=original;
laststate<>adjustcontext );
-->
  modify &G ( type=removenots );
  modify &R ( lastglobal=removenots; laststate=removenots; step=@.step+1 );
};
```

```
--
-- Remove andnots if we are continuing to broaden globally,
-- and the last broaden step was replacing the boolean operators.
--
rule RemoveNotsInit2
{
  &G ( goal type=reformulate );
  &R ( reform  global=narrow: next=broaden; lastlocal=replaceops;
          laststate<>adjustcontext );
-->
  modify &G ( type=removenots );
  modify &R ( lastlocal=removenots; laststate=removenots; step=@.step+1 );
};

--
-- Loosen the boolean operators.
--
--
-- Remove Andnots.
--
rule RemoveNots
{
  &G ( goal type=removenots; subgoal=ll );
  &Q ( query status=evaluated );
-->
local &oldquerynum : integer;
local &newquerynum : integer;
local &name : symbol;
local &oldnum : integer;
local &numpassages : integer;
local &min, &target, &max : integer;
local &decision : symbol;

  &oldnum = &Q.numpassages;
  &min = &G.min;
  &target = &G.justright;
  &max = &G.max;

  &oldquerynum = &Q.version;
  &newquerynum = &oldquerynum + 1;
  call CRemoveAndnots( &oldquerynum, &newquerynum );

  call CPassQueryToMA( &newquerynum, &name );
  &numpassages = CGetNumPassages( &name );

  -- Check that removing andnots doesn't put us outside the acceptable range.
  &decision = CKeepChange( &oldnum, &numpassages, &min, &target, &max );
  if ( &decision = |keep| )
    modify &Q ( status=evaluated; version=&newquerynum;
          numpassages=&numpassages; name=&name );
```

```
  if ( &decision = |discard| )
    call CRemoveQuery( &newquerynum );

  modify &G ( subgoal=whatnext );
};

-- stop reformulation ----------------------------------------------------
--
-- Final Broadening is by adjusting context.
--
rule FinalBroadening1
{
  &G ( goal type=reformulate );
  &R ( reform global=broaden; next=broaden; lastglobal=removenots );
  &Q ( query status=evaluated; context<>26 );
-->
  modify &G ( type=adjustcontext; subgoal=checkcontext );
  modify &R ( laststate=adjustcontext );
};


rule FinalBroadening2
{
  &G ( goal type=reformulate );
  &R ( reform global=narrow; next=broaden; lastlocal=removenots );
  &Q ( query status=evaluated; context<>26 );
-->
  modify &G ( type=adjustcontext; subgoal=checkcontext );
  modify &R ( laststate=adjustcontext );
};


--
-- No More Broadening Possible
--
rule NoMoreBroadening1
{
  &G ( goal type=reformulate );
  &R ( reform global=broaden; next=broaden; lastglobal=removenots );
  &Q ( query status=evaluated; context=26 );
-->
  modify &G ( type=failure );
  remove &R;
  modify &Q ( status=final );
};

rule NoMoreBroadening2
{
  &G ( goal type=reformulate );
  &R ( reform global=narrow; next=broaden; lastlocal=replaceops );
  &Q ( query status=evaluated; context=26 );
-->
```

```
  modify &G ( type=failure );
  remove &R;
  modify &Q ( status=final );
};

--
-- Final Narrowing is by adjusting context.
--
rule FinalNarrowing1
{
  &G ( goal type=reformulate );
  &R ( reform global=narrow; next=narrow; lastglobal=replaceops );
  &Q ( query status=evaluated; context<>20 );
-->
  modify &G ( type=adjustcontext; subgoal=checkcontext );
  modify &R ( laststate=adjustcontext );
};


rule FinalNarrowing2
{
  &G ( goal type=reformulate );
  &R ( reform global=broaden; next=narrow; lastlocal=replaceops );
  &Q ( query status=evaluated; context<>20 );
-->
  modify &G ( type=adjustcontext; subgoal=checkcontext );
  modify &R ( laststate=adjustcontext );
};


--
-- No More Narrowing Possible
--
rule NoMoreNarrowing1
{
  &G ( goal type=reformulate );
  &R ( reform global=narrow; next=narrow; lastglobal=replaceops );
  &Q ( query status=evaluated; context=20 );
-->
  modify &G ( type=failure );
  remove &R;
  modify &Q ( status=final );
};


rule NoMoreNarrowing2
{
  &G ( goal type=reformulate );
  &R ( reform global=broaden; next=narrow; lastlocal=replaceops );
  &Q ( query status=evaluated; context=20 );
-->
```

```
  modify &G ( type=failure );
  remove &R;
  modify &Q ( status=final );
};


-- Goal: reformulate ----------------------------------------------------
-- These rules play with context several times. There is an upper and lower
-- bound, and we iterate until success.

--
-- Adjust context to the query if we want to go in the global direction after
-- having already gone in the local direction.
--
rule AdjustContextInit1 {
  &G ( goal type=reformulate );
  &R ( reform global=@.next; lastlocal<>original );
-->
  modify &G ( type=adjustcontext; subgoal=checkcontext );
  modify &R ( laststate=adjustcontext );
};

--
-- Adjust context to the query if we want to go locally in opposite direction
-- from the global goal, but we've already gone locally as far as we have
-- globally.
--
rule AdjustContextInit2 {
  &G ( goal type=reformulate );
  &R ( reform global<>@.next; lastglobal=@.lastlocal );
-->
  modify &G ( type=adjustcontext; subgoal=checkcontext );
  modify &R ( laststate=adjustcontext );
};

--
-- Once we have begun adjusting the context, continue to do so until success
-- or failure.
--
rule AdjustContextInit3
{
  &G ( goal type=reformulate );
  &R ( reform  laststate=adjustcontext );
-->
  modify &G ( type=adjustcontext; subgoal=checkcontext );
};
```

```
--
-- We want to narrow the query but context is already minimum.
-- Quit with this query as too broad is preferable.
--
rule CannotAdjustContextDown
{
  &G ( goal type=adjustcontext; subgoal=checkcontext );
  &R ( reform next=narrow );
  &Q ( query status=evaluated; context=20 );
-->
  modify &G ( type=failure; subgoal=ll );
  modify &Q ( status=final );
  remove &R;
};

--
-- We want to broaden the query but context is already maximum.
-- Quit with this query.
--
rule CannotAdjustContextUp
{
  &G ( goal type=adjustcontext; subgoal=checkcontext );
  &R ( reform next=broaden );
  &Q ( query status=evaluated; context=26 );
-->
  modify &G ( type=failure; subgoal=ll );
  modify &Q ( status=final );
  remove &R;
};

--
-- We want to narrow the query, but have already tried the narrower context.
-- Quit with the current, broader, context rather than the narrower one.
--
rule DoNotAdjustContextDown
{
  &G ( goal type=adjustcontext; subgoal=checkcontext );
  &R ( reform next=narrow );
  &Q ( query status=evaluated; (@.lo <> @.context) );
-->
  modify &G ( type=failure; subgoal=ll );
  modify &Q ( status=final );
  remove &R;
};

--
-- We want to broaden the query, but have already tried the broader context.
-- Quit with the previous, broader, context rather than the narrower one.
-- This case is handled by just doing the regular AdjustContextUp.
```

```
--
-- We are narrowing/broadening the context by one.
--
rule AdjustContext
{
  &G ( goal type=adjustcontext; subgoal=checkcontext );
-->
  modify &G ( subgoal=changecontext );
};


-- Goal: success ----------------------------------------------------
--
-- We have successfully located passages.
--
rule Success
{
  &G ( goal type=success );
  &Q ( query status=final );
-->
local &version : integer;
local &numpassages : integer;

  &version = &Q.version;
  &numpassages = &Q.numpassages;
  if ( CDisplayOrNo( &version, &numpassages ) = display )
    modify &G ( type=wtpassages )
  else
    modify &G ( type=cleanup );
};

-- Goal: failure ----------------------------------------------------
--
-- We have unsuccessfully located passages.
--
rule Failure
{
  &G ( goal type=failure );
  &Q ( query status=final );
-->
local &version : integer;
local &numpassages : integer;

  &version = &Q.version;
  &numpassages = &Q.numpassages;
  if ( CDisplayOrNo( &version, &numpassages ) = display )
    modify &G ( type=wtpassages )
  else
    modify &G ( type=cleanup );
};
```

```
-- Goal: wtpassages ----------------------------------------------------
--
-- Rank the passages of the name from MICROARRAS.
--
rule WtPassages
{
  &G ( goal type=wtpassages );
  &Q ( query status=final );
-->
local &passage : integer;
local &validpassage : logical;
local &wt : real;
local &name : symbol;
local &displayname : symbol;
local &version : integer;
local &numpassages : integer;

  &name = &Q.name;
  &version = &Q.version;
  &numpassages = &Q.numpassages;
  call CWtPassages( &version, &numpassages, &name, &displayname );
  &validpassage = CGetFirstPassage( &displayname, &passage );
  while (&validpassage)
  {
    &wt = CGetPassageWt( &passage );
    make ( passage status=undisplayed; id=&passage; wt=&wt );
    &validpassage = CGetNextPassage( &displayname, &passage );
  };

  modify &G ( type=displaypassages );
};

-- Goal: displaypassages ----------------------------------------------
--
-- Display the undisplayed passage of highest wt.
-- If there are more than one passage of equal wt, use chronolgical order.
-- Get relevance feedback from the user.
--
rule InteractiveDisplayPassage
{
  &G ( goal type=displaypassages );
  &I ( interactive );
  &P ( passage status=undisplayed );
  ~ ( passage status=undisplayed; wt > &P.wt );
  ~ ( passage status=undisplayed; wt = &P.wt; id < &P.id );
-->
local &passage : integer;

  &passage = &P.id;
  call CDisplayPassage( &passage );
  call CGetRelevance();
  remove &P;
};
```

```
--
-- Display the undisplayed passage of highest wt.
-- If there are more than one passage of equal wt, use chronolgical order.
--
rule DisplayPassage
{
  &G ( goal type=displaypassages );
  ~ ( interactive );
  &P ( passage status=undisplayed );
  ~ ( passage status=undisplayed; wt > &P.wt );
  ~ ( passage status=undisplayed; wt = &P.wt; id < &P.id );
-->
local &passage : integer;

  &passage = &P.id;
  call CDisplayPassage( &passage );
  remove &P;
};

--
-- Remove the displaypassage goal when all passages have been shown.
--
rule DoneDisplayPassage
{
  &G ( goal type=displaypassages );
  ~ ( passage );
-->
local &name:symbol;

  modify &G ( type=cleanup );
};

-- Goal: cleanup ----------------------------------------------------------
--
-- Check for unprocessed queries.
--
rule RemoveUnprocessedQueries
{
  &G ( goal type=cleanup );
  &Q ( query (@.status <> reformulated ∧ @.status <> final) );
-->
  call CErrorHandle( |Unfinished queries in working memory.| );
  remove &Q;
};

--
-- Remove any queries still in working memory.
--
rule RemoveOldQueries
{
  &G ( goal type=cleanup );
  &Q ( query status=reformulated );
-->
```

```
  remove &Q;
};

--
-- Remove any queries still in working memory.
--
rule RemoveFinalQuery
{
  &G ( goal type=cleanup );
  &Q ( query status=final );
-->
local &oldcontext : integer;

  &oldcontext = &Q.context;
  call CChangeContext( &oldcontext, 23 );
  remove &Q;
};


--
-- Remove all concepts in working memory.
--
rule RemoveConcepts
{
  &G ( goal type=cleanup );
  &C ( concept );
-->
local &id : integer;

  &id = &C.id;
  call CRemoveConcept( &id );
  remove &C;
};

--
-- Remove all stems in working memory.
--
rule RemoveStems
{
  &G ( goal type=cleanup );
  &S ( stem );
-->
  remove &S;
};

--
-- Remove any passages still in the system.
--
rule RemovePassages
{
  &G ( goal type=cleanup );
  &P ( passage );
-->
```

```
  call CErrorHandle( |Unprocessed passages in working memory.| );
  remove &P;
};

--
-- Remove any reformulations still in the system.
--
rule RemoveReforms
{
  &G ( goal type=cleanup );
  &R ( reform );
-->
  call CErrorHandle( |Unprocessed reformulations in working memory.| );
  remove &R;
};

--
-- Remove any goals in the system
--
rule RemoveGoals
{
  &G ( goal type=cleanup );
  &G2 ( goal type<>cleanup );
-->
  call CErrorHandle( |Unprocessed goals in working memory.| );
  remove &G2;
};

--
-- Get the next query.
--
rule GetNextQuery
{
  &G ( goal type=cleanup );
-->
  modify &G ( type=getquery );
};


-- Goal: quit ----------------------------------------------------------

--
-- This rule is fired when the only goal left is type=quit.
--
rule Exit
{
  &G ( goal type=quit );
   ~ ( interactive );
-->
  call CKillSystem();
  remove &G;
};
```

```
--
-- This rule is fired when the only goals left are type=quit and
-- the session was interactive.
--
rule InteractiveExit
{
  &G ( goal type=quit );
  &I ( interactive );
-->
  call CKillSystem();
  remove &G;
  remove &I;
};

};
```

## II.3 OMain.ops

This module contains the main procedure.

```
module OMain (opsmain)
{
use OUtils;
use ORules;
use shell;

--
-- opsmain() is the entry point for the program;  It is the first procedure to
-- execute when the program is started.
--
procedure opsmain()
{
local &tracename : symbol;

  call CGetUniqueName( &tracename );
  &TRACE = shellinit( &tracename );
  call toplev( 0, &TRACE );
  call close( &TRACE );
};

};
```

# Appendix III
# Experiment Handouts

## III.1 Query Language

### III.1.1 Query Language (User Only System)

Examples:

dog and (cat or kitten)
*dog and [nextword] *kennel
*dog and *cat andnot [-3 to +3 sentences] bird

Syntax:
term (operator optional_context term ) *

term: word or stemgroup
word: lower case word
stemgroup: *word

operator: **andnot, and, or**

optional_context: [context]
context: n **to** m unit (n < m) or **nextword** or **sentence** or **paragraph**
unit: **words** or **sentences** or **paragraphs**

### III.1.2 Query Language (User + Thesaurus)

Examples:

dog and (cat or kitten)
DOG_KENNEL
*dog and CAT andnot [-3 to +3 sentences] bird

Syntax:
term (operator optional_context term ) *

term: word or stemgroup
word: lower case word
stemgroup: *word or STEMNAME

operator: **andnot, and, or**

optional_context: [context]
context: n **to** m unit (n < m) or **nextword** or **sentence** or **paragraph**
unit: **words** or **sentences** or **paragraphs**

### III.1.3 Query Language (User + Expert System)

Examples:
dog and (cat or kitten)
dog_kennel
dog and cat andnot bird

Syntax:
term (operator term ) *

term: word
word: lower case word

operator: **andnot, and, or**

term: word or phrase
phrase: word(_word)+

No valid contexts! They are under the expert system's control!

## III.2 Questions

### III.2.1 Query Set A

Practice:

What are some sources of error in floating point arithmetic?

Real:

How is computer architecture distinguished from the other
computer design domains?

What are some upward pressures on the level of a machine language?

Fixed length multiplication produces a double length result. How have different machines handled this?

How are interrupts handled? Do not consider techniques to disable them.

### III.2.2 Query Set B

Practice:

I/O devices move. What is the effect of this motion on the architecture of computers?

Real:

What are some design principles that lead to clean architectures?
Do not consider the econonmic advantages of a quality design.

What techniques have been used to reduce bit traffic?

How are control structures implemented?

What role does buffering play in I/O transfers?

### III.2.3 Query Set C

Practice:

Fragmentation of memory is one problem of using a segmentation scheme. How is paging used to fix this?

Real:

Discuss the two fundamentally different ways to formally specify an architecture.

What are the effects of having two zeros, as in the sign magnitude representation of fixed point numbers?

What is done to save state upon a procedure call?

Besides I/O, where is concurrency practiced in the implementation?

## III.3 Questionnaire

Name:

User-only system:
Which techniques did you use for reformulating your queries? Check all appropriate answers.
adjusting context
adding "or" terms - from knowledge, from reading passages
replacing terms - from knowledge, from reading passages
removing "and" terms
adding "andnot" terms
using *terms
whole new query

User-thesaurus system:
Did you use the thesaurus much?
If so, why? If not, why not?
Was it helpful?

User-expert system:
What did you like best?
What did you like least?

Which system was easiest to use?
User-only,
User+thesaurus,
User+expert system

Which system gave the best results?

User-only,
User+thesaurus,
User+expert system

Comments:

# Appendix IV
# Experimental Data

## IV.1 Search performance summarized by system configuration.

### IV.1.1 Number of queries entered.

| Query | User Alone | with Thesaurus | with ES | Mean |
|---|---|---|---|---|
| 1 | 4.50 | 6.00 | 2.25 | 4.25 |
| 2 | 7.75 | 4.50 | 2.25 | 4.83 |
| 3 | 6.75 | 9.75 | 3.75 | 6.75 |
| 4 | 7.00 | 3.25 | 1.75 | 4.00 |
| 5 | 5.00 | 6.00 | 1.25 | 4.08 |
| 6 | 2.75 | 4.75 | 1.00 | 2.83 |
| 7 | 3.75 | 7.00 | 1.50 | 4.08 |
| 8 | 2.25 | 2.50 | 1.00 | 1.92 |
| 9 | 8.00 | 10.00 | 3.25 | 7.08 |
| 10 | 2.75 | 5.50 | 3.75 | 4.00 |
| 11 | 4.25 | 2.25 | 2.75 | 3.08 |
| 12 | 3.25 | 4.00 | 3.75 | 3.67 |
| Mean | 4.83 | 5.46 | 2.35 | 4.22 |

### IV.1.2 Time taken by the user (in seconds).

| Query | User Alone | with Thesaurus | with ES | Mean |
|---|---|---|---|---|
| 1 | 647.75 | 841.75 | 683.75 | 724.42 |
| 2 | 720.00 | 873.75 | 606.25 | 733.33 |
| 3 | 639.00 | 924.25 | 542.75 | 702.00 |
| 4 | 540.75 | 485.75 | 290.75 | 439.08 |
| 5 | 462.25 | 564.75 | 498.75 | 508.58 |
| 6 | 315.75 | 501.75 | 497.25 | 438.25 |
| 7 | 443.75 | 587.00 | 724.00 | 584.92 |
| 8 | 217.25 | 213.50 | 286.75 | 239.17 |
| 9 | 792.00 | 734.25 | 678.50 | 734.92 |
| 10 | 295.00 | 475.50 | 568.75 | 446.42 |
| 11 | 252.25 | 228.25 | 379.25 | 286.58 |
| 12 | 368.00 | 427.00 | 720.25 | 505.08 |
| Mean | 474.48 | 571.46 | 539.75 | 528.56 |

### IV.1.3 Number of relevant passages found (user relevance).

| Query | User Alone | with Thesaurus | with ES | Mean |
|---|---|---|---|---|
| 1 | 8.00 | 10.00 | 5.25 | 7.75 |
| 2 | 4.50 | 6.50 | 9.25 | 6.75 |
| 3 | 4.75 | 6.75 | 4.25 | 5.25 |
| 4 | 9.00 | 14.00 | 7.75 | 10.25 |
| 5 | 6.00 | 4.00 | 7.25 | 5.75 |
| 6 | 7.25 | 5.75 | 7.00 | 6.67 |
| 7 | 3.75 | 6.00 | 8.50 | 6.08 |
| 8 | 9.25 | 7.75 | 9.25 | 8.75 |
| 9 | 8.25 | 5.75 | 5.75 | 6.58 |
| 10 | 4.50 | 5.25 | 4.50 | 4.75 |
| 11 | 9.00 | 6.00 | 10.00 | 8.33 |
| 12 | 14.25 | 14.50 | 8.75 | 12.50 |
| Mean | 7.38 | 7.69 | 7.29 | 7.45 |

IV.1.4 Precision (based on user relevance).

| Query | User Alone | with Thesaurus | with ES | Mean |
|---|---|---|---|---|
| 1 | 0.79 | 0.75 | 0.62 | **0.72** |
| 2 | 0.72 | 0.65 | 0.63 | **0.66** |
| 3 | 0.75 | 0.88 | 0.68 | **0.77** |
| 4 | 0.81 | 0.84 | 0.81 | **0.82** |
| 5 | 0.86 | 0.77 | 0.87 | **0.83** |
| 6 | 0.77 | 0.76 | 0.90 | **0.81** |
| 7 | 0.71 | 0.88 | 0.78 | **0.79** |
| 8 | 0.89 | 0.87 | 1.00 | **0.92** |
| 9 | 0.72 | 0.89 | 0.79 | **0.80** |
| 10 | 0.37 | 0.58 | 0.45 | **0.47** |
| 11 | 0.89 | 0.75 | 0.84 | **0.83** |
| 12 | 0.87 | 0.81 | 0.76 | **0.81** |
| **Mean** | **0.76** | **0.79** | **0.76** | **0.77** |

IV.1.5 Number of relevant passages found (absolute relevance).

| Query | User Alone | with Thesaurus | with ES | Mean |
|---|---|---|---|---|
| 1 | 3.75 | 5.00 | 4.75 | **4.50** |
| 2 | 3.00 | 4.25 | 6.75 | **4.67** |
| 3 | 4.25 | 5.50 | 3.50 | **4.42** |
| 4 | 10.00 | 12.50 | 8.00 | **10.17** |
| 5 | 4.75 | 2.50 | 5.25 | **4.17** |
| 6 | 6.00 | 4.00 | 6.75 | **5.58** |
| 7 | 2.75 | 3.50 | 4.75 | **3.67** |
| 8 | 9.25 | 8.00 | 8.25 | **8.50** |
| 9 | 3.25 | 3.50 | 3.25 | **3.33** |
| 10 | 3.00 | 3.00 | 1.75 | **2.58** |
| 11 | 6.00 | 5.50 | 8.00 | **6.50** |
| 12 | 10.25 | 11.25 | 7.75 | **9.75** |
| **Mean** | **5.52** | **5.71** | **5.73** | **5.65** |

IV.1.6 Recall (based on absolute relevance).

| Query | User Alone | with Thesaurus | with ES | Mean |
|---|---|---|---|---|
| 1 | 0.23 | 0.31 | 0.30 | **0.28** |
| 2 | 0.19 | 0.27 | 0.42 | **0.29** |
| 3 | 0.30 | 0.39 | 0.25 | **0.32** |
| 4 | 0.43 | 0.54 | 0.35 | **0.44** |
| 5 | 0.34 | 0.18 | 0.38 | **0.30** |
| 6 | 0.60 | 0.40 | 0.68 | **0.56** |
| 7 | 0.21 | 0.27 | 0.37 | **0.28** |
| 8 | 0.42 | 0.36 | 0.38 | **0.39** |
| 9 | 0.17 | 0.18 | 0.17 | **0.18** |
| 10 | 0.43 | 0.43 | 0.25 | **0.37** |
| 11 | 0.40 | 0.37 | 0.53 | **0.43** |
| 12 | 0.64 | 0.70 | 0.48 | **0.61** |
| **Mean** | **0.36** | **0.37** | **0.38** | **0.37** |

IV.1.7 Precision (based on absolute relevance).

| Query | User Alone | with Thesaurus | with ES | Mean |
|---|---|---|---|---|
| 1 | 0.32 | 0.40 | 0.57 | 0.43 |
| 2 | 0.38 | 0.41 | 0.44 | 0.41 |
| 3 | 0.52 | 0.71 | 0.62 | 0.62 |
| 4 | 0.83 | 0.78 | 0.84 | 0.82 |
| 5 | 0.74 | 0.52 | 0.62 | 0.62 |
| 6 | 0.70 | 0.55 | 0.87 | 0.71 |
| 7 | 0.35 | 0.52 | 0.43 | 0.43 |
| 8 | 0.74 | 0.90 | 0.89 | 0.84 |
| 9 | 0.30 | 0.54 | 0.47 | 0.44 |
| 10 | 0.19 | 0.28 | 0.14 | 0.20 |
| 11 | 0.65 | 0.69 | 0.68 | 0.67 |
| 12 | 0.62 | 0.63 | 0.68 | 0.64 |
| Mean | 0.53 | 0.58 | 0.60 | 0.57 |

IV.2 Search Performance summarized by subject.

IV.2.1 Number of Queries entered by the user.

| Subject | User Alone | with Thesaurus | with ES | Mean |
|---|---|---|---|---|
| 1 | 3.75 | 2.75 | 2.75 | 3.08 |
| 2 | 7.75 | 6.50 | 5.75 | 6.67 |
| 3 | 2.75 | 5.75 | 1.50 | 3.33 |
| 4 | 5.00 | 8.75 | 3.25 | 5.67 |
| 5 | 6.50 | 5.50 | 1.50 | 4.50 |
| 6 | 5.00 | 2.25 | 2.50 | 3.25 |
| 7 | 3.50 | 7.25 | 3.75 | 4.83 |
| 8 | 5.75 | 4.50 | 2.25 | 4.17 |
| 9 | 5.00 | 5.00 | 1.00 | 3.67 |
| 10 | 1.75 | 6.75 | 1.75 | 3.42 |
| 11 | 6.75 | 5.00 | 1.25 | 4.33 |
| 12 | 4.50 | 5.50 | 1.00 | 3.67 |
| Mean | 4.83 | 5.46 | 2.35 | 3.81 |

IV.2.2 Time taken by the user.

| Subject | User Alone | with Thesaurus | with ES | Mean |
|---|---|---|---|---|
| 1 | 358.50 | 261.50 | 706.00 | 442.00 |
| 2 | 683.75 | 776.75 | 866.25 | 775.58 |
| 3 | 343.00 | 544.50 | 328.50 | 405.33 |
| 4 | 434.75 | 577.75 | 471.00 | 494.50 |
| 5 | 651.75 | 380.75 | 512.00 | 514.83 |
| 6 | 318.50 | 251.00 | 481.75 | 350.42 |
| 7 | 566.00 | 1104.50 | 670.25 | 780.25 |
| 8 | 377.50 | 316.00 | 476.25 | 389.92 |
| 9 | 450.75 | 767.25 | 409.25 | 542.42 |
| 10 | 152.50 | 648.00 | 470.25 | 423.58 |
| 11 | 893.50 | 520.25 | 419.00 | 610.92 |
| 12 | 463.25 | 709.25 | 666.50 | 613.00 |
| Mean | 474.48 | 571.46 | 539.75 | 549.17 |

IV.2.3 Number of relevant passages found (user relevance).

| Subject | User Alone | with Thesaurus | with ES | Mean |
|---|---|---|---|---|
| 1 | 6.75 | 4.5 | 7.5 | 6.25 |
| 2 | 6.75 | 6 | 4.75 | 5.83 |
| 3 | 6 | 9.5 | 7.25 | 7.58 |
| 4 | 12 | 7.25 | 6.75 | 8.67 |
| 5 | 7.5 | 9.25 | 8.75 | 8.50 |
| 6 | 5.5 | 5.75 | 10.5 | 7.25 |
| 7 | 6.75 | 7.75 | 6.5 | 7.00 |
| 8 | 6.25 | 5.5 | 4.75 | 5.50 |
| 9 | 8.25 | 10.75 | 8 | 9.00 |
| 10 | 7.25 | 8.5 | 7.5 | 7.75 |
| 11 | 6.5 | 8.25 | 7.75 | 7.50 |
| 12 | 9 | 9.25 | 7.5 | 8.58 |
| Mean | 7.38 | 7.69 | 7.29 | 7.94 |

IV.2.4 Precision (based on user relevance).

| Subject | User Alone | with Thesaurus | with ES | Mean |
|---|---|---|---|---|
| 1 | 0.58 | 0.88 | 0.77 | 0.74 |
| 2 | 0.37 | 0.74 | 0.51 | 0.54 |
| 3 | 0.68 | 0.78 | 0.70 | 0.72 |
| 4 | 0.71 | 0.69 | 0.72 | 0.71 |
| 5 | 0.93 | 0.89 | 0.91 | 0.91 |
| 6 | 1.00 | 0.97 | 0.96 | 0.98 |
| 7 | 0.76 | 0.75 | 0.66 | 0.72 |
| 8 | 0.86 | 0.57 | 0.47 | 0.63 |
| 9 | 0.73 | 0.73 | 0.94 | 0.80 |
| 10 | 0.94 | 0.68 | 0.77 | 0.79 |
| 11 | 0.78 | 0.90 | 0.91 | 0.86 |
| 12 | 0.82 | 0.85 | 0.79 | 0.82 |
| Mean | 0.76 | 0.79 | 0.76 | 0.83 |

IV.2.5 Number of relevant passages found (absolute relevance).

| Subject | User Alone | with Thesaurus | with ES | Mean |
|---|---|---|---|---|
| 1 | 4.75 | 4.25 | 6.50 | 5.17 |
| 2 | 8.25 | 4.00 | 3.75 | 5.33 |
| 3 | 5.50 | 7.00 | 5.75 | 6.08 |
| 4 | 7.00 | 5.00 | 4.25 | 5.42 |
| 5 | 5.50 | 5.50 | 7.25 | 6.08 |
| 6 | 2.75 | 4.75 | 5.25 | 4.25 |
| 7 | 7.25 | 6.75 | 6.00 | 6.67 |
| 8 | 5.25 | 6.25 | 5.75 | 5.75 |
| 9 | 6.00 | 8.00 | 6.00 | 6.67 |
| 10 | 4.75 | 5.75 | 6.50 | 5.67 |
| 11 | 4.50 | 5.75 | 6.00 | 5.42 |
| 12 | 4.75 | 5.50 | 5.75 | 5.33 |
| Mean | 5.52 | 5.71 | 5.73 | 5.47 |

IV.2.6 Number of relevant passages (absolute relevance).

| Subject | User Alone | with Thesaurus | with ES | Mean |
|---|---|---|---|---|
| 1 | 4.75 | 4.25 | 6.50 | 5.17 |
| 2 | 8.25 | 4.00 | 3.75 | 5.33 |
| 3 | 5.50 | 7.00 | 5.75 | 6.08 |
| 4 | 7.00 | 5.00 | 4.25 | 5.42 |
| 5 | 5.50 | 5.50 | 7.25 | 6.08 |
| 6 | 2.75 | 4.75 | 5.25 | 4.25 |
| 7 | 7.25 | 6.75 | 6.00 | 6.67 |
| 8 | 5.25 | 6.25 | 5.75 | 5.75 |
| 9 | 6.00 | 8.00 | 6.00 | 6.67 |
| 10 | 4.75 | 5.75 | 6.50 | 5.67 |
| 11 | 4.50 | 5.75 | 6.00 | 5.42 |
| 12 | 4.75 | 5.50 | 5.75 | 5.33 |
| **Mean** | **5.52** | **5.71** | **5.73** | **5.47** |

IV.2.7 Recall (based on absolute relevance).

| Subject | User Alone | with Thesaurus | with ES | Mean |
|---|---|---|---|---|
| 1 | 0.36 | 0.31 | 0.38 | 0.35 |
| 2 | 0.46 | 0.25 | 0.25 | 0.32 |
| 3 | 0.39 | 0.37 | 0.41 | 0.39 |
| 4 | 0.45 | 0.31 | 0.24 | 0.33 |
| 5 | 0.28 | 0.36 | 0.53 | 0.39 |
| 6 | 0.15 | 0.33 | 0.34 | 0.27 |
| 7 | 0.47 | 0.41 | 0.44 | 0.44 |
| 8 | 0.40 | 0.45 | 0.34 | 0.39 |
| 9 | 0.45 | 0.43 | 0.43 | 0.44 |
| 10 | 0.31 | 0.47 | 0.36 | 0.38 |
| 11 | 0.27 | 0.40 | 0.43 | 0.37 |
| 12 | 0.38 | 0.31 | 0.40 | 0.36 |
| **Mean** | **0.36** | **0.37** | **0.38** | **0.37** |

IV.2.8 Precision (based on absolute relevance).

| Subject | User Alone | with Thesaurus | with ES | Mean |
|---|---|---|---|---|
| 1 | 0.44 | 0.80 | 0.72 | 0.65 |
| 2 | 0.43 | 0.46 | 0.41 | 0.43 |
| 3 | 0.63 | 0.56 | 0.58 | 0.59 |
| 4 | 0.39 | 0.47 | 0.50 | 0.45 |
| 5 | 0.56 | 0.52 | 0.77 | 0.62 |
| 6 | 0.46 | 0.76 | 0.39 | 0.54 |
| 7 | 0.82 | 0.67 | 0.58 | 0.69 |
| 8 | 0.49 | 0.58 | 0.58 | 0.55 |
| 9 | 0.52 | 0.50 | 0.72 | 0.58 |
| 10 | 0.59 | 0.43 | 0.67 | 0.56 |
| 11 | 0.60 | 0.59 | 0.70 | 0.63 |
| 12 | 0.41 | 0.56 | 0.61 | 0.53 |
| **Mean** | **0.53** | **0.58** | **0.60** | **0.57** |

**IV.3 Ranking summarized by user and query.**

**IV.3.1 Random ranking expected (queries versus subjects).**

| Query | S1 | S2 | S3 | S4 | S5 | S6 | S7 | S8 | S9 | S10 | S11 | S12 | Mean |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 5.50 | | | 5.00 | | | | 6.33 | | 3.00 | | | 4.96 |
| 2 | 6.50 | | | 5.50 | | | | 9.00 | | 7.00 | | | 7.00 |
| 3 | 3.50 | | | 3.33 | | | | 3.00 | | 3.50 | | | 3.33 |
| 4 | 4.25 | | | 4.00 | | | | 5.00 | | 8.00 | | | 5.31 |
| 5 | | | | | 5.00 | | | | 4.50 | | 3.00 | 6.00 | 4.63 |
| 6 | | | | | 4.50 | | | | 4.50 | | 4.00 | 4.50 | 4.38 |
| 7 | | | | | 6.50 | | | | 5.00 | | 6.50 | 4.50 | 5.63 |
| 8 | | | | | 5.00 | | | | 5.00 | | 5.00 | 5.50 | 5.13 |
| 9 | | 3.25 | 6.00 | | | 4.00 | 2.67 | | | | | | 3.98 |
| 10 | | 2.50 | 3.25 | | | 4.50 | 7.00 | | | | | | 4.31 |
| 11 | | 5.25 | 5.25 | | | 4.00 | 3.50 | | | | | | 4.50 |
| 12 | | 5.00 | 2.25 | | | 7.50 | 7.50 | | | | | | 5.56 |
| Mean | 4.94 | 4.00 | 4.19 | 4.46 | 5.25 | 5.00 | 5.17 | 5.83 | 4.75 | 5.38 | 4.63 | 5.13 | 4.89 |

**IV.3.2 Ranking produced by expert system (queries versus subjects).**

| Query | S1 | S2 | S3 | S4 | S5 | S6 | S7 | S8 | S9 | S10 | S11 | S12 | Mean |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 4.42 | | | 4.45 | | | | 4.74 | | 1.71 | | | 3.83 |
| 2 | 6.30 | | | 4.80 | | | | 6.40 | | 4.80 | | | 5.57 |
| 3 | 3.76 | | | 3.64 | | | | 3.40 | | 3.55 | | | 3.59 |
| 4 | 4.54 | | | 4.67 | | | | 5.15 | | 7.95 | | | 5.58 |
| 5 | | | | | 4.58 | | | | 4.07 | | 3.17 | 5.77 | 4.40 |
| 6 | | | | | 4.25 | | | | 4.29 | | 3.36 | 4.45 | 4.09 |
| 7 | | | | | 9.13 | | | | 5.08 | | 6.05 | 2.50 | 5.69 |
| 8 | | | | | 5.00 | | | | 5.00 | | 4.56 | 5.67 | 5.06 |
| 9 | | 3.13 | 5.33 | | | 3.67 | 2.75 | | | | | | 3.72 |
| 10 | | 1.67 | 1.00 | | | 4.15 | 4.20 | | | | | | 2.75 |
| 11 | | 6.08 | 4.89 | | | 4.23 | 3.39 | | | | | | 4.65 |
| 12 | | 6.33 | 2.10 | | | 6.79 | 4.69 | | | | | | 4.98 |
| Mean | 4.75 | 4.30 | 3.33 | 4.39 | 5.74 | 4.71 | 3.76 | 4.92 | 4.61 | 4.50 | 4.29 | 4.59 | 4.49 |